De Montfort University

An Investigation into Induction Motor Vector Control Based on Reusable VHDL Digital Architectures and FPGA rapid Prototyping

Abdulmagid Aounis

September 2002

A thesis submitted in partial fulfillment of the requirements of

De Montfort University

For the degree of Doctor of Philosophy

ABSTRACT

The research investigates a novel approach, based on reusable VHDL digital architectures, to induction motor vector control modelling, simulation and FPGA rapid prototyping. Several induction motor control methods are reviewed. The *dq-axis* vector control strategy is established as the most appropriate technique for achieving decoupled control of the flux and torque producing components of the induction motor stator current. The space phasor theory is presented and the principles of rotor flux oriented vector control are outlined.

Most a.c. drives in use today adopt a microprocessor/DSP based digital control strategy, but implementation of the current loop and PWM control are still tied to analogue control circuitry. Although these control schemes have the advantage of fast dynamic response, they suffer the disadvantages of circuit complexity, limited functions and difficulties when circuit modification is required. Recent developments in the field of microelectronics have enabled complex switching strategies for transistorized inverters to be implemented by means of ASIC technology. Consequently, motor control and power conversion systems employing ASIC/FPGA technology are receiving increased attention.

A comprehensive study of the modelling, simulation and design of a vector controlled induction motor using VHDL hardware description language and targeting FPGA implementation is presented. Initially, the complete drive, including the induction motor, power electronics and the controller is behaviourally modelled using VHDL and then simulated and evaluated. Speed and torque responses of the motor to various input conditions, like step or variable inertia loads are considered.

The Clark_transform, Park_transform, Current_model, PI_controller, Control_unit, Inverse Park_transform and PWM Waveform Generator blocks are decribed and their hierarchical design to implement an induction motor vector control strategy is examined. The VHDL design, synthesis, timing simulation and Xilinx FPGA implementation of the induction motor digital controller is described. The universal applicability (reusability) of the control blocks to a wide range of vector control drives, with different topologies, is shown to be a major advantage. It is also shown that the employment of FPGAs for commissioning trial provides further benefits such as: rapid prototyping, simple hardware / software design and compact implementation.

The complete digital controller circuit design is synthesized, implemented and comprehensively tested. The results confirm correct controller operation in conjunction with the interface circuitry, the power system and the induction motor. The new hardware description language based modelling technique, based on hierarchical design and VHDL modular development, is validated by both simulation results and experimental tests.

ACKNOWLEDGEMENTS

I would like to extend my deep gratitude to my supervisor, Professor Malcolm McCormick, for his invaluable guidance and support throughout the course of my research. I am also greatly indebted to Dr. Marcian. N. Cirstea for his excellent advice both as a second supervisor and as a friend.

I will always be grateful to Dr. A. Dinu, Dr. J. G. Khor, Dr.Yan-Ting Hu, my colleague A. Abouarkoub and Adel Faraon for their help with information and advice concerning the practical experiments.

Special thanks to Mrs Veena Vora and Mrs Sheila Hayto, secretaries of the faculty, for their help and advice proven to be an invaluable asset in many situations.

Many thanks are due to others academics at DMU and to the technical support and administrative staff, especially Dilip Chauhan, Tim O'mara, Steve Regester, Manbir Sambhi and Pritesh Karia.

I wish to thank my parents for their unfailing support and encouragement. Last, but not in any way least, I am particularly grateful to my wife and my children Hana, Shima, Doa and my son Abdulbaray for their faith in me and their patience during all these years.

CONTENTS

1.0 INTRODUCTION 1

1.1 OVERVIEW OF THE THESIS 4

2.0 POWER ELECTRONICS – A REVIEW 6

2.1 THE DEVELOPMENT OF POWER ELECTRONICS 8

2.2 ELECTRIC MOTORS 9

2.2.1 Market Information 9

2.3 BASIC PRINCIPLE OF MICROPROCESSOR/MICROCOMPUTER CONTROL 10

2.4 ASICS AND FPGAS 12

3.0 INDUCTION MOTOR CONTROL STRATEGIES 15

3.1 VECTOR CONTROL 17

3.1.1 Current Control 18

3.1.2 Hysteresis Control 19

3.1.3 Model Reference Adaptive Control System (MRACS) 23

3.1.4 Direct Torque and Flux Control (DTFC) 24

3.1.5 Sliding Mode Control (SMC) 26

4.0 MODERN CONTROL SYSTEMS DESIGN USING CAD TECHNIQUES 30

4.1 TRENDS IN CONTROL SYSTEM 30

4.2 TRADITIONAL CONTROL 32

4.3 MICROCOMPUTER CONTROL 33

4.4 DIGITAL SIGNAL PROCESSING CONTROL (DSP) 33

4.5 FUZZY LOGIC CONTROL (FLC) AND ARTIFICIAL NEURAL NETWORK (ANN) 35

4.6 ELECTRONIC DESIGN AUTOMATION (EDA)/CAD TECHNIQUES 36

4.6.1 Application Specific Integrated Circuits (ASIC 's) 38

4.6.2 Field-Programmable Gate Array (FPGAs) 40

4.6.2.1 FPGA Structure 40

4.6.3 VHSIC Hardware Description Language (VHDL) 42

4.6.3.1 Abstraction Levels and HDL's 43
4.6.3.2 Structural Design 43
4.6.3.3 Behavioural Design 44
4.6.3.4 Physical Design 44
4.6.4 Design Flow in Top Down Methodology 45
4.7 XILINX FOUNDATION SERIES 47
4.8 THE ADVANTAGES OF VHDL 49
5.0 SYSTEM MODELLING AND SIMULATION 51
5.1 THE SPACE VECTOR MODEL OF THE A.C. MACHINE. 52
5.1.1 The 3-Phase (abc) to 2-Phase ($\alpha\beta$) Transformation 53
5.1.2 Commutator Transformation. 57
5.1.3 Transformation of a Rotating 3-Phase (abc) Quantity to Stationary 2-Phase (dq) Quantity. 58
5.2 MODELLING AND SIMULATION OF THE INDUCTION MOTOR. 60
5.2.1 Induction Machine Model. 60
5.2.1.1 Electrical Model 60
5.2.1.2 Mechanical Model 61
5.2.1.3 Vector Control Modelling 61
5.2.1.4 Induction Machine Complete Model 62
5.3 VHDL FUNCTIONAL SIMULATION. 63
5.4 THE ROTOR FLUX ORIENTED VECTOR CONTROL 70
5.4.1 The Flux Model. 72
5.4.2 Control Strategy 75
5.4.2.1 Pulse Width Modulation 75
5.4.2.2 Space Vector Modulation 77
5.4.2.3 The FOC AlgorithmStructure 78
5.4.3 Simulation Results. 79
6.0 VHDL DESIGN OF THE FIELD ORIENTED CONTROLLER (FOC) 92
6.1 THE 3 AXIS – 2 AXIS CURRENTS TRANSFORMATION IN VHDL 93
6.2 THE (α, β) TO (d, q) PROJECTION (PARK TRANSFORMATION) 99
6.2.1 Control Unit Description 101
6.2.2 The Multiplier 102

6.2.3 Registers *105*
6.3 CURRENT MODEL 109
6.4 THE DIVIDER 114
6.5 GENERATION OF SINE AND COSINE VALUES 116
6.6 OVERALL SIMULATION OF THE CURRENT MODEL 118
6.7 PI CONTROLLERS 119
6.7.1 Zero Order Hold (ZOH) *121*
6.7.2 First Order Hold (FOH) *122*
6.7.3 PI Regulators *123*
6.8 PWM WAVEFORM GENERATOR 127
6.9 THE OVERALL SIMULATION OF THE CONTROLLER 134

7.0 EXPERIMENTAL TESTS 138

7.1 HARDWARE DESIGN PROCESS 138
7.2 GENERAL IMPLEMENTATION ISSUES 139
7.3 FPGA IMPLEMENTATION AND TESTING OF THE VECTOR CONTROL SYSTEM 141
7.3.1 The Implementation of the Clark Transform *142*
7.3.2 The Implementation of the Park Transform *145*
7.3.3 The Implementation of the Multiplier *150*
7.3.4 The Implementation of the Divider *154*
7.3.5 The PWM Control Signals Output Block *156*
7.4 COMPLETE SYSTEM SETUP AND TEST 159
7.4.1 Clocking Circuit *166*
7.4.2 Overall experiment test *170*

8.0 CONCLUSIONS AND FURTHER WORK 174

8.1 CONCLUSIONS 174
8.2 ORIGINAL CONTRIBUTION TO THE THESIS 177
8.3 FURTHER WORK 178
8.3.1 Embedded System *178*
8.3.2 Sliding Mode Control *179*

REFERENCES 182

PUBLICATIONS 191

APPENDIX A A1

APPENDIX B B1

APPENDIX C C1

APPENDIX D D1

APPENDIX F F1

Glossary of symbols

T_{max} maximum rated torque (Nm).

T_e the electromagnetic torque (Nm).

ω_{max} maximum rated speed.

ϑ_e flux position.

ω_r angular speed of the motor (rads/s).

ϑ_r the rotor angle which is the angle between the magnetic axes of the stator winding and the rotor winding (rads).

ω_{mr} speed of the rotor flux linkage space phasor (rads/s).

ρ_r phase angle of the rotor flux linkage space phasor with respect to the direct axis of the stator reference frame (rads).

ω_{sl} angular slip frequency (rads/s).

i_a, i_b, i_c phase currents of the motor.

i_{sx}, i_{sy} the instantaneous values of the direct and quadrature axis stator current components respectively and expressed in the general reference frame (A).

$|\bar{i}_{mr}|$ module space phasor of the rotor magnetizing current expressed in the magnetizing flux oriented reference frame (A).

T_s, T_r stator and rotor time constant (s).

T_s' stator transient time constant (s).

L_s' stator transient inductance (H).

R_s, R_r resistance of a stator and rotor phase winding respectively (Ohm).

J moment of inertia ($Kg.m^2$).

L dynamic inductance (H).

L_m magnetizing inductance (H).

L_r, L_s rotor and stator inductance respectively (H).

P	number of poles.
$\bar{i}_s$	space phasor of the stator current expressed in the stator reference frame (A).
$\bar{\psi}_{r\psi r}$	space phasor of the rotor flux linkages expressed in the rotor flux oriented reference frame (Wb).
ψ_{rx}, ψ_{ry}	instantaneous values of direct and quadrature axis rotor flux linkage components respectively and expressed in the general reference frame (Wb).
v_{dx}, v_{dy}	instantaneous values of direct and quadrature axis decoupling voltage components expressed in the general or special reference frame (V).
v_{sx}, v_{sy}	instantaneous values of direct and quadrature axis stator voltage components respectively and expressed in the general or special reference frame (V).
i_{sd},i_{sq}	d-q axis stator current in the stationary reference frame (A).
i_{qr},i_{dr}	d-q axis rotor current in the stationary reference frame (A).
e	back e.m.f voltage (V).
$\lvert \Delta \bar{i} \rvert$	magnitude of the current error vector (A).

Subscript

*	reference.
r	rotor.
s	stator.
F's	represent voltages, currents and fluxes.
PI	Proportional and Integral controller.
h	hysteresis band.

Chapter 1

Introduction

Before the advent of power semiconductor devices, a.c. machines were commonly considered for fixed speed applications, the speed being dictated by the frequency of the supply voltage while d.c. motors were considered the workhorses in industry for variable speed applications. Although the control principles and the converter equipment required for d.c. drives is simple, the d.c. machine has disadvantages, for example it is expensive when compared to the simple and rugged cage type induction motor, the need for commutators and brushes make it unreliable and unsuitable for operating in dusty and explosive environments and it requires frequent maintenance [1],[2]. While the a.c. machine is less expensive, more efficient and reliable, especially the cage type induction motor, the cost of the converter and the control is considerably higher, making the a.c. drive more expensive than the d.c. drive. In addition, the control of a.c. drives is very complex and needs intricate signal processing to obtain the comparable performance of d.c. drives. Present technology aims to provide substantial cost reduction and performance improvement of a.c. drive systems.

With the availability of fast switching and more easily controlled power electronic components such as the Gate Turn Off thyristor (GTO), the Bipolar Junction Transistor (BJT) and the IGBT, research engineers realised that new control schemes could be implemented for a.c. machines. Several control algorithms were developed in the late 1960's by which a.c. induction motors could almost achieve torque control. However, these methods met with only limited success due to the fact that too many complicated calculations involving many unknown model parameters and numerous approximations need to be performed. However in the 1970's a new algorithm for a.c. induction motor control, known as Field Oriented Control was introduced by F. Blaschke [3]. In this approach a two vector method is applied to the induction motor by separating the stator current into a flux-producing component and an orthogonal torque-producing component. Thereby, field orientation provides the same de-coupled control of torque and flux as with the d.c. machine.

Subsequently advanced techniques, incorporating vector control, have been implemented in induction machine drive systems. The objective is to improve performance by developing fast acting controllers, which react to load disturbances and input reference variations as fast as d.c. machine systems. Such dynamical systems require to be studied on a transient basis and consequently modelling of induction machines using generalized machine theory is employed when predicting performance.

Recent developments in the area of microprocessors has enabled digital implementation of complex Pulse Width Modulation (PWM) schemes to be achieved. Two basic schemes are used to operate the inverter motor assembly: open-loop and closed-loop control schemes. Open-loop control is the simplest form of control which essentially supplies the voltage and frequency expected to produce the correct speed and torque. Closed-loop control is more sophisticated control strategy, providing a range of characteristics, typically accurate speed control and controlled slip, to enable reduced energy consumption at low torque.

Vector control is a modern closed-loop control scheme. The term vector control is derived from the fact that in an a.c. machine both the phase angle and the modulus of the current can be controlled. Vector control is a technique that promises high performance a.c. drives with control characteristics equal to the best d.c. drives. To achieve satisfactory systems it is necessary to control both the flux-producing and torque-producing components of the current. In a d.c. motor this is possible because the field and armature currents are distinct

and therefore can be individually controlled. However, in an induction motor since both components of current are supplied through the three phase stator, these cannot be easily identified and controlled.

The application of vector control techniques in a.c. drives demands accurate position and speed feedback information for the current control and servo-control loops. In many applications, the rotor position signal is obtained from a mechanical sensor, such as an optical encoder or a resolver, that may reduce the system's reliability and add significantly to the drive costs. Consequently, a strong interest arises in the alternative sensorless control, using only stator voltage and current measurement based on state observers. State observers are usually implemented to reconstruct the inaccessible states of the controlled process.

PWM inverters are the most used power converters for IM supply. Various current control techniques for Pulse Width Modulated (PWM) inverters have been investigated and reported in the literature (e.g. [4], [5], [20]). These are dealt with in detail in chapter 2.

It is well known that the control of induction machines is fraught with complexities and technological challenges. Simple controllers exhibit non linear characteristics and, in general, operate satisfactory only over a limited speed range. Variable frequency control involves the use of an inverter connected to the machine and, until recently, the design of the inverter and machine system were separate activities. However, recently the trend has been towards the integration of the design process thereby facilitating the study of the effects of the inverter output waveform on the dynamic load and vice versa. To do this effectively, appropriate models of both systems are a pre-requisite.

1.1 Overview of the Thesis

The aim of the proposed research is to investigate different types of control techniques in order to facilitate the design of an universal controller for power converters. Initially the power system is modeled using VHDL and then the controller is simulated and evaluated. The design, synthesis and timing simulation of the system is achieved using VHDL, comprehensively described by Perry [6] and Navabi [60], and XILINX software [7]. Fast XILINX FPGAs are used for implementation. Comprehensive tests were carried out. The main objectives to be achieved within the stated aim of this research work are:

- To investigate a range of vector control strategies for d.c. link fed induction motors.
- To develop an innovative model (behavioural-functional modelling) of a complete vector control scheme for the speed control of an induction motor using VHDL.
- To simulate the novel functional models of different modules in the control algorithm and to achieve overall simulation of the VHDL model of the complete drive system. This will lead to the achievement of a new methodology of IM vector control prototyping.
- The VHDL modular and hierarchical design for synthesis and implementation of the vector control system, leading to the development of a complete set of reusable VHDL modules for IM vector control.
- Synthesis, Timing Analysis (Simulation) and evaluation of the each reusable VHDL module and of the complete controller design.
- To implement and download the controller into a fast XILINX FPGA using VHDL and XILINX Foundation Software.
- Comprehensive experimental testing and evaluation of the novel controller.
- Complete Vector controlled Induction motor drive set-up, experimental testing and evaluation.

The approach adopted for the system representation combines the vector control strategy with a new modelling and design technique. The new approach uses Very High Speed Integrated Circuit Hardware Description Language (VHDL), one of the most popular standard HDLs. It is supported by all major Computer Aided Engineering (CAE) platforms and synthesis tools can compile VHDL designs into a large variety of target technologies.

The complete system is modelled and simulated in VHDL and then the digital controller is designed using VHDL. The controller was then synthesised and downloaded into a XILINX FPGA (XCS40PQ208) for rapid prototyping. The VHDL approach presented provides important advantages such as: wide compatibility of the design with respect to different CAE software tools (VHDL files are ASCII files), a large range of implementation technologies and the reuse of the VHDL code.

The thesis is divided into eight chapters, including this introductory chapter. Chapter 2 briefly examines the development of power electronics and electric motors from the time of their invention to the present. Chapter 3 briefly traces the work recently published on induction motor control. Particular attention is devoted to the work concerned with vector control strategies applied to induction motor. Various induction motor speed control methods are considered. Chapter 4 introduces the modern control systems design using EDA/CAD techniques.

Chapter 5 describes a new approach to the modelling, simulation and the controller design of a complete induction motor electric drive system. The novel technique uses a hardware description language (VHDL) as a unique EDA environment for the system modelling, evaluation and controller design. Simulation results are presented, proving the validity of the model for the vector controlled induction motor system. The induction motor model is presented together with the VHDL code used for modelling. To prove the validity of the model, various simulations are carried out, the corresponding results are plotted and a few commentaries are given. Chapter 6 describes a new approach to the design of a complete digital induction motor electric drive system using synthesisable VHDL codes and also presents the architecture and the operation of the motor controller. The practical test results are presented and discussed in chapter 7. In the final chapter of the thesis, chapter 8, there is a general discussion on the research project and conclusions are drawn. Possible further developments of the research work are outlined.

Chapter 2

Power Electronics – A Review

This chapter briefly outlines the development of power electronic devices and electric motors from the time of their invention to the present. Since the invention of electrical machines in the 19th century, there has been a need to convert electrical power for various applications such as electrical machine drives, voltage regulation and industrial processes, for example welding and heating. Initially, rotating machines were predominantly used to control and convert electrical power. It was the introduction of the glass bulb mercury arc rectifier (1900), which led to the beginning of the power electronics era. Power Electronics is the branch of engineering concerned with the application of electronics in the control and conversion of electrical power.

The expression, "power electronics", refers to both power electronics and control. Broadly, "power" deals with the static and rotating power equipment for the generation, transmission, and distribution of electric power, "electronics" deals with the solid state devices and circuits for signal processing to meet the desired control objectives [8] and "control" deals with the steady state and dynamical characteristics of closed loop systems. With the development of power semiconductor technology, the power handling capacity and the switching speed of the power devices has improved tremendously.

Several types of power semiconductor devices have been developed and are commercially available, such as: power thyristors, GTOs, FCTs, BJTs, FETs, IGBTs and power diodes. For the control of electric power conditioning, the conversion of electric power from one form to another is necessary, and the switching characteristics of the power devices allow these conversions. The static power converter performs the power conversion function. A converter may be considered as a switching matrix and can be classified into six types:

- ♦⇒ *Diode rectifiers.*
- ♦⇒ *a.c* - *d.c* converters (controlled rectifiers).
- ♦⇒ *a.c* - *a.c* converter (*a.c* voltage controller).
- ♦⇒ *d.c*-*d.c* converters (*d.c* choppers).
- ♦⇒ *d.c* - *a.c* converter (inverter).
- ♦⇒ Static switches.

The development of microprocessor technology has had a great impact on the synthesis of control strategies for power semiconductor devices. Similarly, over the past four decades, automatic control has been greatly enhanced by the corresponding developments in electronics, particularly in digital computer technology.

The low cost and advances in digital signal processing power have now taken a.c. drives to the point where all the signal processing required for high dynamic performance can be executed by a single microprocessor implemented on a postcard sized printed circuit board. The main advantage of microprocessor digital control is that the same standard hardware can be used for many different applications because the functioning of the processor is determined by flexible, individually designed software. With the modern advancement of power electronics, induction motor drives are even more widely used in many industrial applications. A literature review on each topics has been carried out to provide relevant background and is presented in the next section.

2.1 The Development of Power Electronics

Early power electronic devices such as thyratrons and ignitrons were crude and unreliable. In 1948, the invention of the p-n junction transistor by Bardeen, Brattian and Shockley (Bell Laboratories) was a revolutionary advancement in the field of electronics. This laid the foundation for the development of the p-n-p-n transistor switch by J. L. Moll, *et. al.* (1956), a device which later became known as the thyristor, or silicon controlled rectifier (SCR). By 1957, the first commercial thyristor was made available by General Electric Company. This three terminal device had a continuous current rating of 25A and a blocking voltage of up to 300V. This marked the beginning of the modern power electronics era.

Another class of power electronic devices subsequently developed was the controllable power switch. Although controllable switches like the transistor have been around since 1948, designing them to posses high power handling capabilities was not achieved until much later. Examples of devices in this category are Gate Turn-off Thyristors (GTOs), power transistors, power MOSFETs, Integrated Gate-Commutated Thyristors (IGCTs) and Insulated Gate Bipolar Transistor (IGBTs). The GTO is a thyristor-like latching device that can be turned off by a negative gate current.

Power transistors and power MOSFETs were developed from small signal designs to later versions capable of handling voltage of the order of hundreds of volts. In the early 1980s, the IGBT was developed [9], This combines the low on-state conduction losses of the Bipolar Junction Transistor (BJT) and the high switching frequency of the MOSFET. The IGBT has since gained widespread popularity in power electronic applications. Developments in semiconductor technology have resulted in power electronic devices becoming smaller, reliable, more efficient, less expensive and having higher power ratings.

2.2 Electric Motors

Electric motors are major users of electricity in industrial plants and commercial premises. Motive power accounts for almost half of the total electrical energy used in the UK and nearly two thirds of industrial electricity use. Among all the existing motors on the market there are three classical motors as illustrated in Figure 2.1.

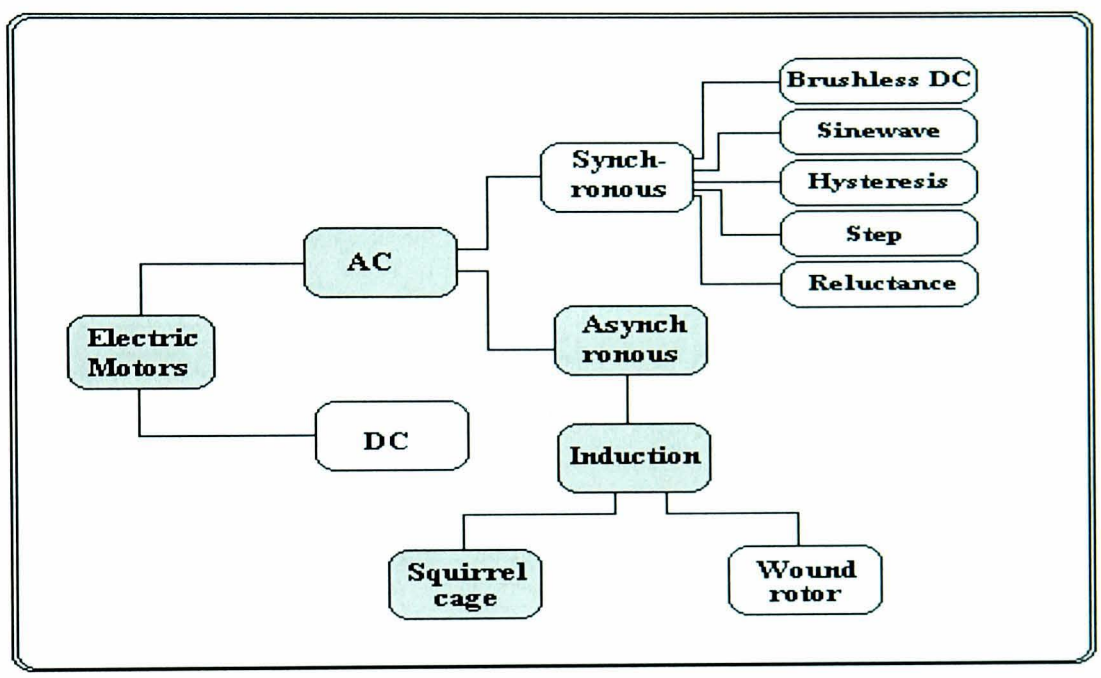


Figure 2.1 - *Electric motors classifications*

The polyphase induction motor, invented by Nikola Tesla in 1886, is the most widely used and the least expensive industrial motor. The stator winding is connected to the a.c. source, and the rotor winding is either short-circuited, as in squirrel-cage machines, or closed through external resistance, as in wound-rotor machines. Squirrel cage machines are also known as brushless machines and wound rotor machines are termed slip-ring machines.

2.2.1 Market Information

In the mid 1980s the U.S. market for a.c. drives was over $250 million and grew to $500 million by 1992. Nowadays it is estimated to be over $700 million. More than ten million motors, with a total capacity of 70 GW, are installed in UK industry alone.

With regard to the induction motor market, an estimated 50 million or more of them are now in use in American industry, totalling some 150 million horsepower. Each normal year's production adds about a million motors to that number. In addition, another 20 million single-phase fractional-horsepower induction motors are used in domestic appliances such as electric fans, refrigerators, washing machines, and farm appliances [10].

Advancing a.c. drive performance, cost optimization, application flexibility and the environmental needs for energy efficiency are the prime driving forces for a.c. drive market growth. Although a.c. drives are penetrating some d.c. drive applications, most of the a.c. growth stems from new applications which previously had no adjustable drive control. The d.c. drive market will continue to sustain itself due to retrofit opportunities on existing d.c. drives. New a.c. vector drives which presently encompass about 10% of the market will satisfy the largest portion of the a.c. market growth in the coming years. Volts/hertz a.c. drives, still retain a strong presence in the marketplace especially for stand alone applications and also for systems requiring multiple motors on single drive. Volts/hertz a.c. drives are widely accepted as being less sophisticated than vector drives, easier to set-up and are cost effective [10]. The range and typical application area for the most common type of electric drive, the induction motor are reviewed in chapter 3.

2.3 Basic Principle of Microprocessor/Microcomputer Control

The range and typical application of microprocessor and microcomputer control are briefly reviewed. Digital controllers have been used to give results as good as, or better than, analogue controllers in numerous cases, with the added feature that the control strategies can be varied by simply reprogramming the computer instead of having to change the hardware. In addition, analogue controllers are susceptible to ageing and drifting, which in turn cause degradation in performance. These advantages have attracted many users to adopt digital technology in preference to conventional methods and caused computer control to be applied to many areas. Some of the current interest areas are: auto-pilots for aero-planes satellite altitude control, industrial and process control, robotics, navigational systems and radar, building energy management and control.

With advances in VLSI (Very Large Scale Integration) and denser packing capabilities, faster integrated circuits can be manufactured which result in quicker and more powerful computers. Therefore, application to control areas which a few years ago were considered to be impractical or impossible because of computer limitations, are now entering the realms of possibility.

Traditional control systems are normally implemented using analogue and digital hardware. In its relatively short existence, digital computer technology has touched, and had a profound effect upon, many areas of life. Its enormous success is due largely to the flexibility and reliability that computer systems offer to potential users. This, coupled with the ability to handle and manipulate vast amounts of data quickly, efficiently and repeatedly, has made computers extremely useful in many varied applications. In control systems the digital computer acts as the controller and provides the enabling technology that allows the design and implementation of the overall system, so that satisfactory performance is obtained.

Digital control systems differ from continuous ones in that the computer can only act at instants of time rather than continuously. This is because a computer can only execute one operation at a time, and so the overall algorithm proceeds in a sequential manner. Hence, taking measurements from the system and processing them to compute an activating signal, which is then applied to the system, is a standard procedure in a typical control application.

A recent advance in computer systems is in the area of parallel processing, where the computational task is shared out between several processors that can communicate with each other in an efficient manner. Individual processors can solve sub-problems, with the results being brought together in some ordered way, to arrive at the solution to the overall problem. Since many processors can be incorporated to execute the computations, it is possible to solve large and complex problems quickly and efficiently.

One of the problems in a computer control system is the interfacing between computers and continuous systems so that the analogue plant signals can firstly be read into the computer, and then digital control signals can be applied to the system. Analogue signals must be converted into digital form for analysis in the computer, and the digital signals from the computer have to be converted back to analogue form for application to the plant under control.

The superiority of microcomputer control over conventional hardware based control can be recognized for complex drive control systems. The simplification of hardware improves system reliability and reduces cost. Particularly important in many applications is that digital control provides noise immunity, which is particularly important here because of large power switching transients in the converters. The complex computation and decision taking capabilities of microcomputers enables the application of the modern optimal and adaptive control theories to optimize the drive system performance. In addition, powerful diagnostic routines can be written in the software. Microcomputer technology is advancing at such a fast rate that the use of efficient high-level language with large hardware integration and VLSI implementation of the controller is easily possible.

Unlike dedicated hardware control, a microcomputer executes control in serial fashion, i.e. multitasking operations are performed in time multiplexed method. As a result, the slow computation capability may pose serious problem in executing the fast control loops. However, the problem can be solved by multi-microprocessor control, where judicious partitioning of tasks can significantly enhance the execution speed. These are some basic aspects of microcomputer/microprocessor control.

2.4 ASICs and FPGAs

Presently, many digital control systems are microprocessor based, primarily because of the availability of control Integrated Circuits (ICs), cheaper memories and tremendous advancements in data handling capabilities. However Application Specific Integrated Circuits (ASICs), which successfully replace microprocessors by using modern Computer Aided Design (CAD) / Electronic Design Automation (EDA) techniques have revolutionized control schemes in many applications. FPGAs are a special flexible class of ASICs, allowing programming by the end user. Xilinx [11], one of the major manufacturers of FPGAs, defines them as " high density Application Specific Integrated Circuits (ASICs), combining the logic integration of custom Very Large Scale Integration (VLSI) with the time to market and cost advantages of standard products".

Another major FPGA producer, Lucent Microelectronics [12], stresses the versatility of the FPGA and the time to market benefits. Initially FPGAs were mostly looked at as being of importance for the rapid prototyping of circuits, which would later be hardwired and thus being a design tool to aid profitability, rather than being of use directly in applications. As chips with higher gate counts and as faster routing as well as placing tools becoming available this view has changed.

The advantages of being able to place a large number of functions onto a single chip and being able to reconfigure this chip right on the desktop, and thus remove the non recurring engineering costs from prototyping and testing of new designs, is presented by Small [13]. Utilizing high level programming languages allows a complete picture of structure and functionality of the whole circuit to be built up via a text file that is then fed through the appropriate software to produce a layout of gates, known as a netlist, and downloaded into the FPGA.

The importance of integrated system design, where software/hardware elements are not viewed as totally separate but all tasks are considered as being potentially carried out by either, is constantly being stressed in modern design theory. Significant problems with testing systems before design - particularly the hardware elements due to the paucity of totally reliable "hardware simulators" available for many processors, is commented on by Hoffman in [14].

This is where the main advantages of a system based around FPGAs for rapid prototyping of AISCs is seen. These allow hardware and software to be rebuilt and both can be tweaked and debugged in the same way. Recently, the arrival of full program debugging tools such as break points and single stepping for FPGAs, alongside specialised tools such as waveform viewers, have allowed the full implementation of software/hardware developments. Hardware Description Languages circuit entry tools have significant advantages in allowing the designer to break the design task down into palatable chunks - much as software designers do. This is often referred to as "divide and conquer". Today FPGAs are likely to have as big an impact on the computing world as programmable microprocessors did when they were first introduced [14].

As has been already noted, custom chips have of course been around a long time, but prior to the advent of FPGAs they represented a huge financial investment.

A direct comparison between the implementation of the same design into hardware on an FPGA, and into software, using a Pentium processor, is described by Page in [15]. A program for determining the greatest common divisor was written and had its performance measured in a variety of situations. The code is written in such a way so that it can be compiled either for downloading onto hardware or by an ordinary C-compiler for implementation on the Pentium.

The program was compiled for four different environments, three were FPGA based and one was a Pentium 300. The results showed that the Pentium 300 ran approximately three times faster than the quickest FPGA but as the FPGA clock speed was only 14 MHz compared to 300MHz of the Pentium the power consumption and heat generation of the FPGA is considerably lower. This example illustrates the great save in power consumption enabled by the use of FPGA for digital circuits implementation.

The IP method allows companies to sell standard solutions written in HDL for implementation on the designer's own FPGAs and thus remove an element of "re-inventing the wheel", while spreading development time, and thus costs, around different companies. This mimics the process in early microprocessor development, where standard functions were implemented through the purchase of TTL integrated circuits.

The FPGA is increasingly being seen as a system on a chip solution, especially for large and growing number of applications, such as control system design, video frame buffer processing, internet data encryption and laser printer engines. In addition, many of these new applications themselves make use of existing IP cores, for example encryption or Fast Fourier Transform algorithms. Utilising these algorithms, time to market for new FPGA implementations can be vastly reduced over the need both to design and produce a new chip or even to fully code a new solution from scratch.

Chapter 3

Induction Motor Control Strategies

The following section discuss the control system of induction motors, including topics such as Vector Control, Hysteresis control, Direct Torque Control, Model Reference Adaptive Control and Sliding Mode Control theory, to relate these developments to the present research work.

High performance control and estimation techniques for induction motor drives constitutes a very fascinating and challenging subject. Recently they received wide attention in the literature. With the invention of vector or field oriented control and the demonstration that a.c. motors can be controlled like separately excited d.c. motors, a high control performance is now finding increasing acceptance in industrial drives for applications, such as steel mills, paper mills, servos, machine tools, robotics, elevators, and electric vehicle. The availability of efficient power switches and fast processors has facilitated the growth in development and use of induction motor drives. In a typical induction motor drive Figure 3.1, the power converter serves to convert the supply energy into a form suited for the operation of the motor. The output characteristic of the power converter can be controlled to appear like an adjustable magnitude and frequency current or voltage source to the motor.

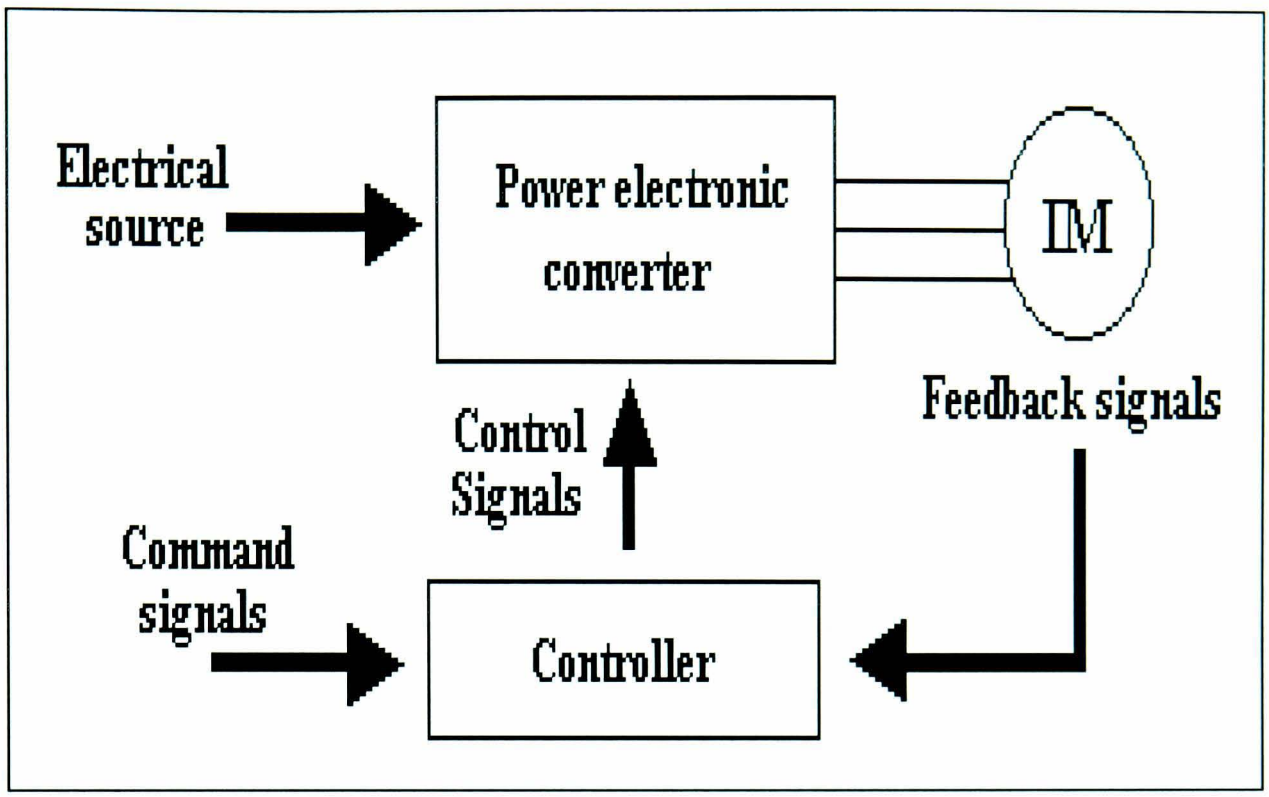


Figure 3.1 - *Block diagram of an induction motor drive*

Induction motor control strategies can be classified as scalar control and vector control strategies. The scalar control operates utilising simplified equations derived from the general space vector model. This approach involves only the space vector amplitudes and their corresponding frequencies and the simplified equations are valid only in steady state region. Consequently, scalar control is simple but generates poor response during transient operation [26]. In contrast, vector control operates directly with the space-vector model of the motor, therefore, it offers good results in both steady-state and transient operation. For any speed control algorithm, the most important information is the actual motor speed. There are two possible approaches to get the speed information: using a speed sensor or calculating the speed based on the electrical motor quantities. These two methods are applicable to scalar control as well as to vector control, but the use of vector control ensures better dynamic response of the drive system. Improved controllers and control strategies have led to increased use of a.c. drives in new applications where variable speed, with good speed control and fast acting response to load changes is required.

3.1 Vector Control

Vector control strategies can be used to vary the speed of an induction motor over a wide range. In the vector (also called field oriented) control scheme, a complex current is synthesized from two quadrature components, one of which is responsible for the flux level in the motor, while the other controls torque production in the motor. Essentially, the control problem is reformulated to resemble d.c. motor control. Vector control offers a number of benefits including speed control over a wide range, precise speed regulation and fast dynamic response. There are two ways of achieving induction machine vector control: one is direct vector control, which implies that the rotor flux vector is directly sensed by detectors, for example Hall probes or calculated from the stator currents and voltages. This approach is attractive in theory, yet when practically applied it attracts high cost plus possible estimation errors at low speed. Alternatively an indirect method can be used. In this method the rotor flux vector is estimated from stator currents and the rotor speed. The major deficiency of this approach is the sensitivity to motor parameter variation cause by magnetic saturation and temperature changes. The basic control elements of both direct and indirect control are essentially the same, however the unit vector ρ_e that transforms the synchronously rotating stator voltages into stationary frame signals which is derived from the flux vector that can be estimated either by voltage model, current model, or by close loop observer. It is also possible to estimate the speed signal from the voltage and current signals. Kreindler et al [16] and Bose [17] showed that the direct field orientation is more advantageous than the indirect type of control as it overcomes the controller sensitivities due to motor parameter variations.

In order to fulfill the objectives of this research and to design an universal controller for a.c. motors, different types of control strategies are reviewed, such as:

- Current Control.
- Hysteresis Control.
- Model Reference Adaptive Control.
- Direct Torque Control.
- Sliding Mode Control.

3.1.1 Current Control

In current controlled PWM inverters which are widely applied to high performance a.c. drives and reactive power compensation systems current control plays the most important role. Various current control strategies have been developed in recent years [18, 19, 20, 21]. Easy implementation, fast dynamic response, maximum current limit, and insensitivity to load parameter variation are the desirable characteristics sought. Among the techniques considered and most often used is a voltage source inverter VSI-PWM system with current controller, as illustrated in Figure 3.2.

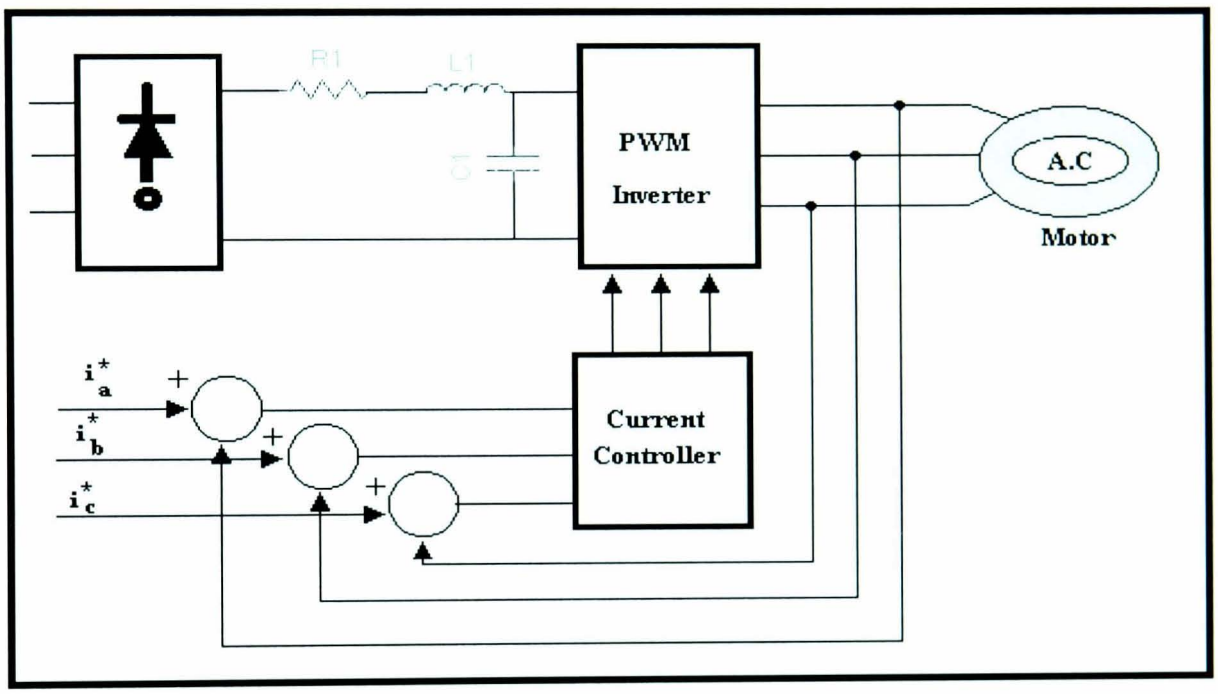


Figure 3.2 - *Basic system diagram of PWM current controller*

Current controllers can be classified as hysteresis, ramp comparison, or predictive controllers. The ramp comparison controllers compare the current errors to a triangle waveform to generate the inverter firing signals. Predictive controllers calculate the inverter voltages required to force the currents to follow the current references [22].

The Hysteresis controller is reviewed in greater depth because of its simplicity and widespread use.

3.1.2 Hysteresis Control

Hysteresis controllers utilise some type of hysteresis in the comparison of the line currents to the current reference [18], [23]. A space vector based hysteresis current regulator has been proposed in [20]. This regulator is able to suppress the high harmonic current under steady state conditions and provide a rapid current response in transient state but it still uses an estimation of the counter EMF based on the derivative of high frequency current components.

Another two simple space vector current control strategies are proposed in by Marian. P et al [23]. Both are based on three level hysteresis comparators which select appropriate inverter voltage regulators via switching EPROM table. A schematic is shown in Figure 3.3.

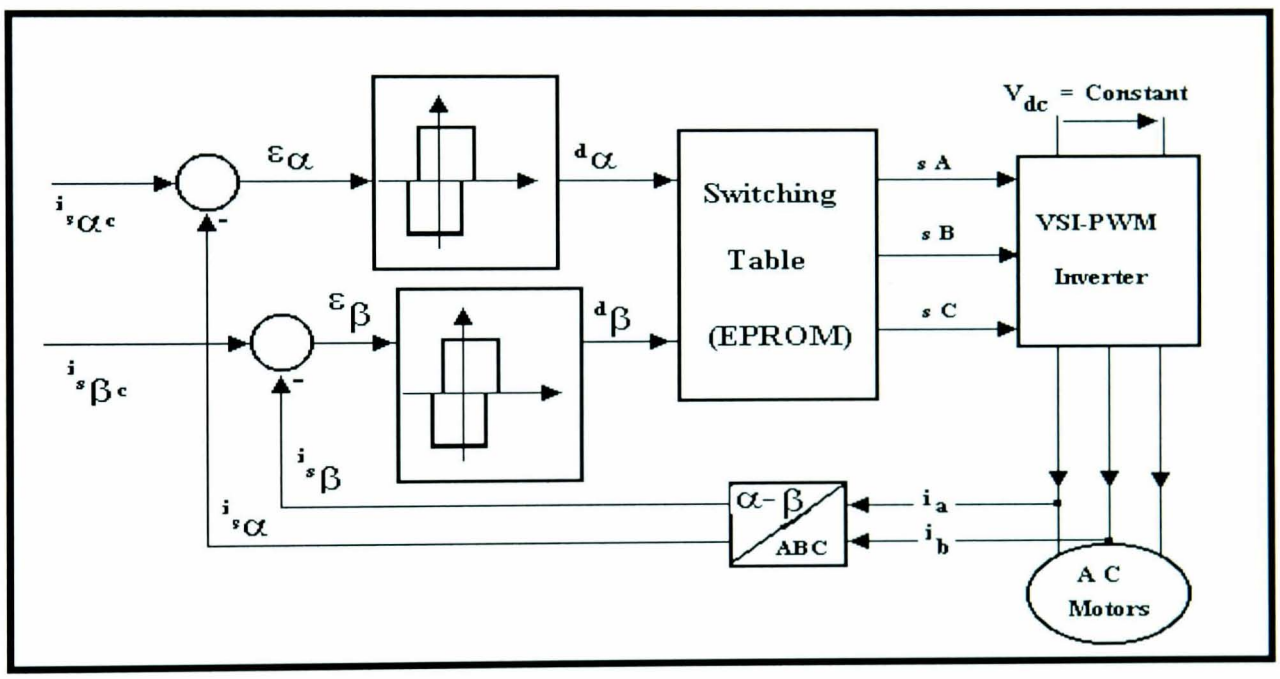


Figure 3.3 - *Block diagram of current vector controller*

This guarantees a precise voltage vector selection. The proposed current vector controller working in α-β co-ordinate is shown in Figure 3.3. The system forms an output current space vector according to $i_{s\alpha c}$ and $i_{s\beta c}$ commands. The accuracy of the current formation is determined by the width of the hysteresis zone of three level comparators. The second controller proposed works with current components represented in a rotated (field_oriented) co-ordinates system (x-y). In this controller the field oriented stator current commands i_{sxc} and i_{syc} as well as the slip frequency command are input quantities of the system.

The instantaneous current errors are delivered to the hysteresis comparators. For the current control strategy proposed it is enough to analyse only signs of voltage vector component. The strategy for the current control used is based on the following statements:

If $\varepsilon_\alpha > \Delta H$ ***THEN*** $d_\alpha = 1$ ***and*** $u_{s\alpha} = u$ $(u_{s\alpha} > 0)$.

If $\varepsilon_\alpha < -\Delta H$ ***THEN*** $d_\alpha = 0$ ***and*** $u_{s\alpha} = -u$ $(u_{s\alpha} < 0)$.

The output of the hysteresis (d_x *and* d_y) and the rotor flux position create a digital word, which by accessing the address of the EPROM obtains the set of switching pulses (S_A, S_B, S_c).

A novel Space-Vector-Modulation (SVM) based Hysteresis Current Controller (HCC) is presented by Kwon, Kim and Youm [24].This technique utilizes all the advantages of the HCC and SVM technique. The controller determines a set of space vectors from a region detector and applies a space vector selected according to the main HCC. A set of space vectors including the zero vector to reduce the number of switching is determined from the sign of the output frequency and output signals of the three comparators with a little larger hysteresis band than that of the main HCC. The SVM technique confine space vectors to be applied according to the region where the output voltage vector is located. To obtain the zero output current error, the SVM technique requires a measurement of the counter emf vector which is not practical. On the other hand HCC can be utilized to make the output current vector track the command vector with almost negligible response time and insensitivity to line voltage and load parameter variations. The configuration has the advantages of reducing significantly the number of switchings than the conventional HCC and at the same time gives the same space vectors as those obtained from SVM technique.

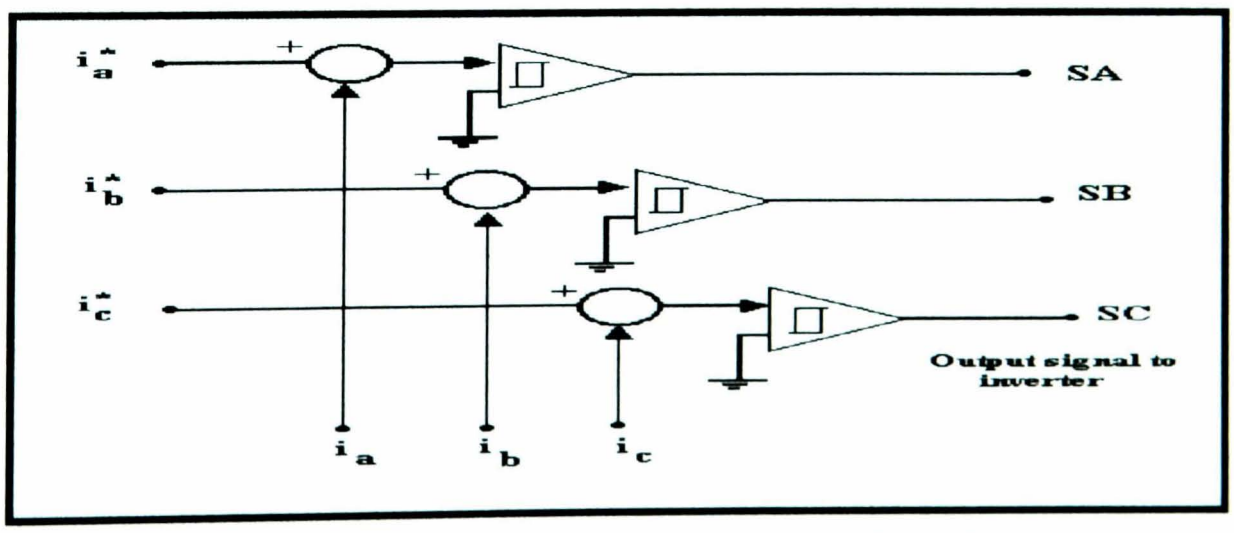


Figure 3.4 - *Hysteresis current control scheme using independent comparators*

A typical three phase hysteresis current controller shown in Figure 3.4 [22]. The controller works on the principal of detecting the motor currents and comparing them with the references using three independent hysteresis comparators having a hysteresis band h. The comparators output signals are used to produce the drive signals for the inverter power switches. The state of the switches depends on the comparison results. The motor phase is connected to the positive terminal of the d.c. source if the current in this phase is lower than $i_{ref}-h/2$ and connected to the negative terminal of the dc source if the current is higher than $i_{ref} + h/2$. This control scheme is simple and characterized by good dynamic performance.

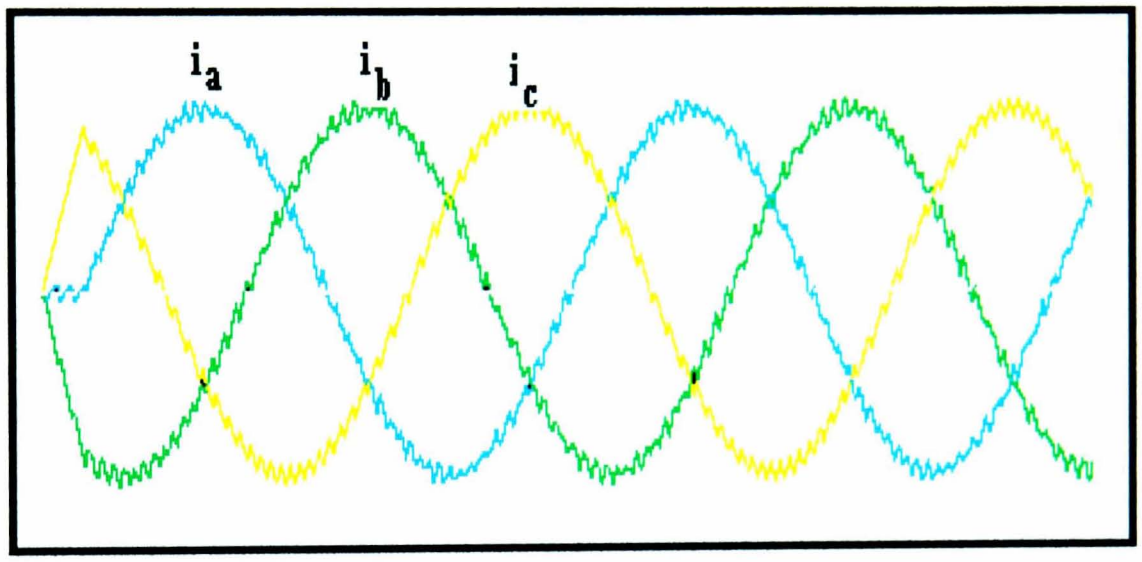


Figure 3.5 - *Line current waveform*

The performance of the proposed control scheme was simulated and the result is shown in Figure 3.5. The concept of voltage and current space vectors is used to simplify the notation in representing a set of three phase voltages and currents. The inverter voltage space vector is defined as a combination of the phase voltages v_a ,v_b and v_c:

$$V = \frac{2}{3}(v_a + av_b + a^2 v_c). \qquad \textbf{2-9}$$

Where $a = e^{j2\pi/3}$

The inverter current space vector is defined in a similar manner as

$$i = \frac{2}{3}(i_a + ai_b + a^2 i_c). \qquad \textbf{2-10}$$

The function of the current controllers is to force the motor current vector i to follow as closely as possible the reference current vector i^*. This can be accomplished by comparing the actual motor currents and the references. The inverter power switches are then activated in such a manner that reduces the current error vector $\Delta i = (i^* - i)$ to a minimum.

The current control techniques proposed in [20, 23, 24] have the following features and advantages : hardware implementation is very simple and practically free of adjustment; they need only two hysteresis comparators and an EPROM table; no motor counter emf is required. With the information located in the EPROM table and the on line information from the hysteresis comparators, the realisation of very simple but intelligent current controllers with the characteristics shown in Figure 3.6 and Figure 3.7 can be realised.

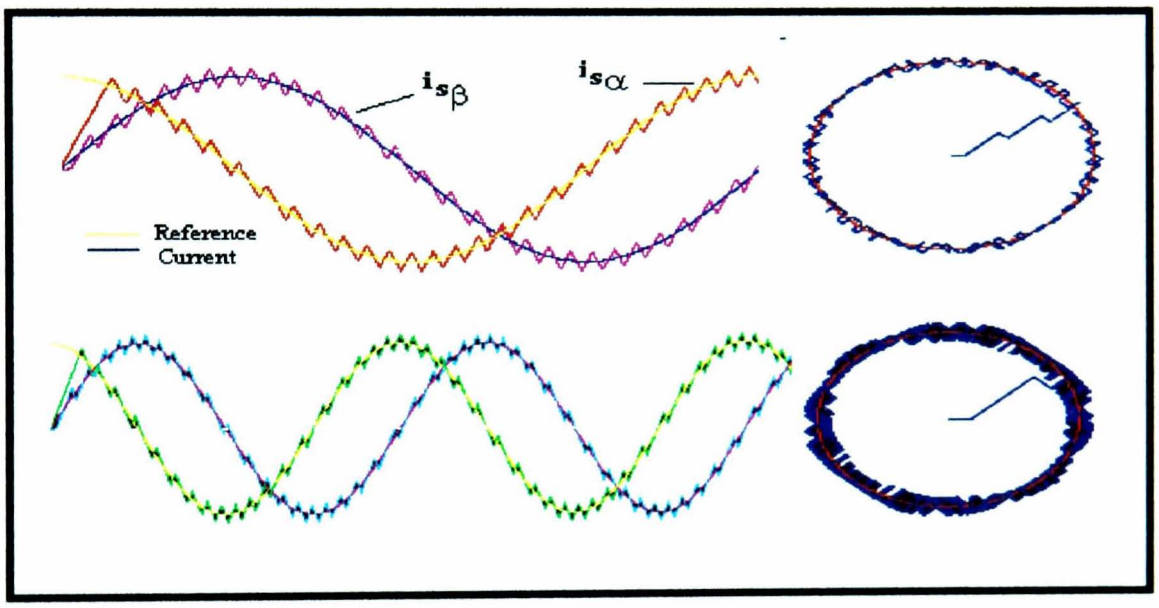


Figure 3.6 - *Simulated waveforms for the proposed methods*

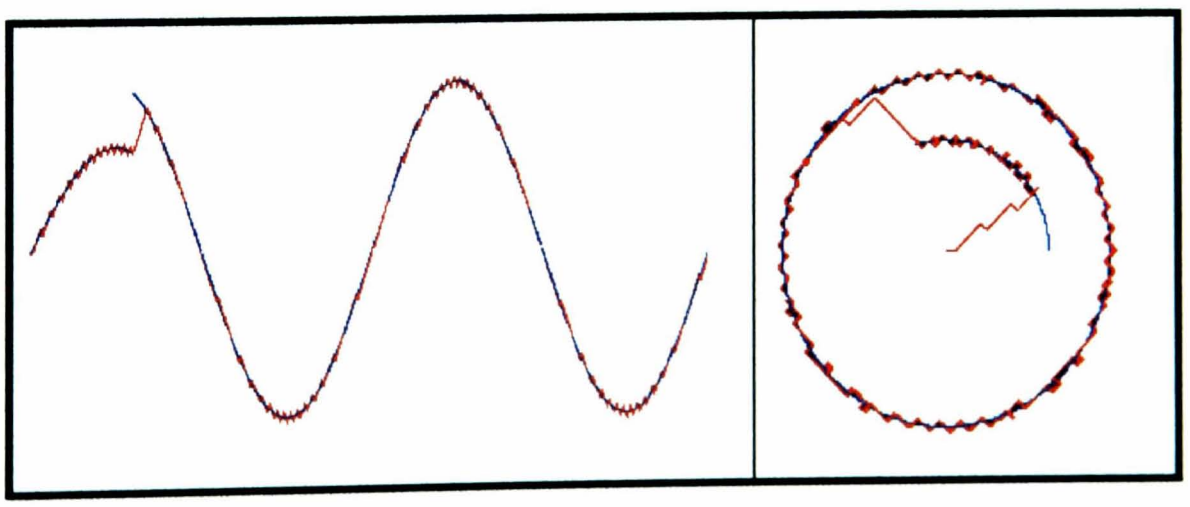


Figure 3.7 - *Simulation of the dynamic performance response to a step change of reference current for hysteresis control*

High performance motion control with induction motors requires separate control of the flux and the current producing the torque. As flux measurement in induction motors is difficult, the flux is often indirectly controlled by controlling an intermediate variable, which is usually the *d-axis* Park component of the stator current. Due to its dependence on the motor model, this flux control method is naturally sensitive to parameter uncertainties and requires advanced control strategies which allows an on line parameter identification. This can be achieved using a model reference adaptive control system.

3.1.3 Model Reference Adaptive Control System (MRACS)

Advanced vector control for induction motor drives is analysed in [25, 26, 27]. The on line parameter identification is achieved by an application of the Model Reference Adaptive System *(MRAS)* theory Figure 3.8.

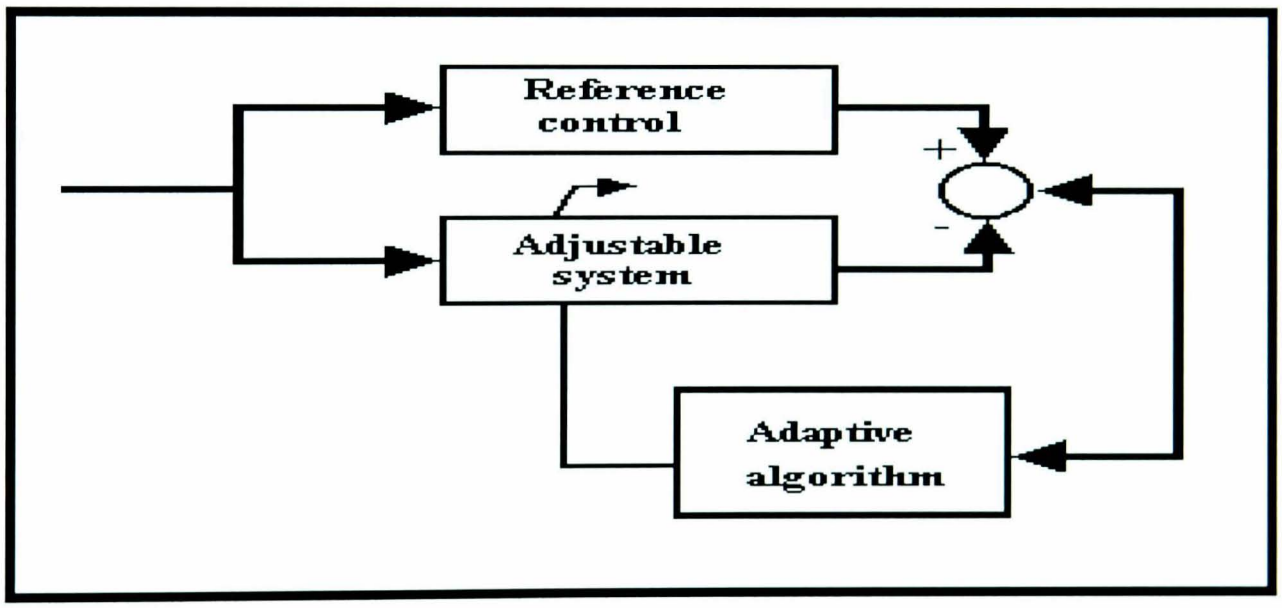


Figure 3.8 - *Model reference adaptive system*

MRAS is a type of adaptive system that can be utilized for rotor speed identification in the rotor flux oriented control system of a voltage source inverter fed induction machine. The adjustable system corresponds to the mathematical model (rotor speed estimation model) of the vector controlled induction machine whose speed is driven by the rotor speed adaptation algorithm which implements rotor speed identification.

The reference model specifies the desired rotor speed ω_r and in the adjustable model the speed to be estimated $\hat{\omega}_r$ is an unknown parameter. The output of the reference control model is compared to the output of the adjustable model and the difference is used by the adaptive algorithm to modify the rotor speed of the adjustable system so that the error between the output of the reference model and that of the adjustable model approaches zero.

An other type of vector control algorithms that ensures fast transient response and generates simple implementations due to the absence of the closed loop current control is the Direct Torque Control (DTC)

3.1.4 Direct Torque and Flux Control (DTFC)

The *DTFC* method shown in Figure 3.9 is basically a performance enhanced scalar control method and is popularly known as Direct Torque Control *(DTC).*

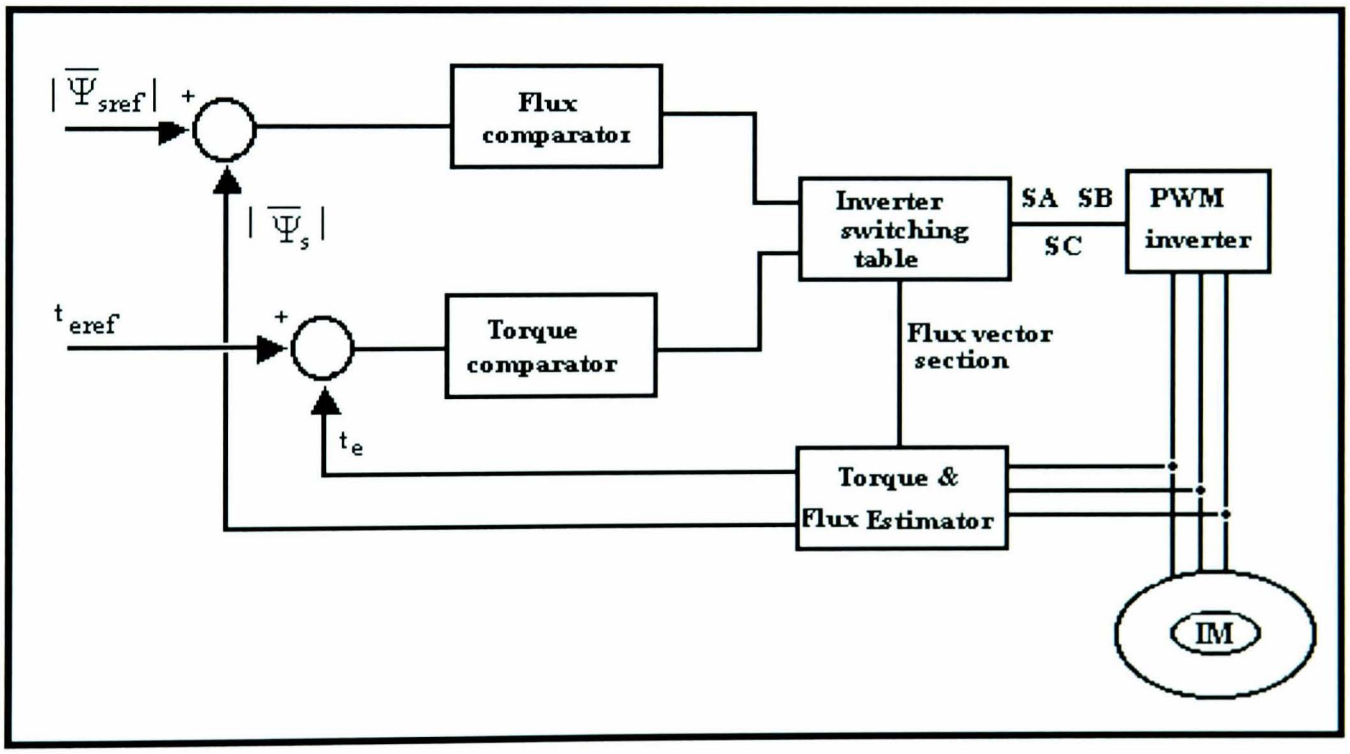


Figure 3.9 - *Schematic of DTC induction motor control*

DTC was developed by German and Japanese investigators more than 15 years ago [28] for use in the torque control of high power servo drives. Recently, in some applications, it has provided an alternative to the field oriented control strategy. In *DTC* the stator flux magnitude is controlled within a hysteresis band.

Basically, *DTC* has torque and stator flux control in the outer loops, as indicated in Figure 3.9. The machine voltages and currents are sensed to estimate the torque and flux vector that gives information about the stator flux location in one of the 60 degree sectors. The control loop errors then are used to generate the digital signal through the respective hysteresis band comparators. A three dimensional look up table then selects the most appropriate voltage vector to satisfy the flux and torque demands. Since the feedback signals are estimated from the machine terminal, the low speed performance limitation and parameter variation problem are similar to the stator flux oriented direct vector control method. The theory behind *DTC* is based on expressing the motor stator currents i_a i_b and i_c by the stationary frame *d-q* quantities as:

$$i_{ds} = i_a \qquad \textbf{2-11}$$

$$i_{qs} = (i_a + 2i_b) \qquad \textbf{2-12}$$

Likewise, the stator *d-q* flux λ_{ds}, λ_{qs} and stator *d-q* voltage v_{ds}, v_{qs} is obtained from the three phase values. The stator flux can be written in terms of the stator voltages and the stator *iR* drop as:

$$\lambda_{ds} = \int_0^t (v_{ds} - i_{ds} R_s) dt \qquad \textbf{2-13}$$

$$\lambda_{qs} = \int_0 (v_{ds} - i_{qs} R_s) dt \qquad \textbf{2-14}$$

Thus the change in the stator flux in a switching period T_s can be approximated by:

$$\Delta\lambda_s = (v_s - i_s R_s) T_s \approx V_s T_s \qquad \textbf{2-15}$$

Hence the electromagnetic torque produced by the machine is given by:

$$T = \frac{3}{2}\frac{P}{2}(\lambda_{ds} i_{qs} - \lambda_{qs} i_{ds}) \qquad \textbf{2-16}$$

Where P is the pole number of the machine.

The main advantages of this technique is the simplicity and the digital form of the control structure [28, 29, 30, 31].

A type of control system that has received the recent attention of several researchers in the area of power electronics and variable speed drives is the Sliding Mode Control (SMC).

3.1.5 Sliding Mode Control (SMC)

SMC control strategies for the induction motor are proposed in [30, 32, 33, 34, 35, 36]. It is also known as the Variable Structure Control system *(VSC)*. In sliding mode control, the reference model is stored in the form of a predefined phase plane trajectory, and the drive system response is forced to follow or slide along the trajectory by a switching control algorithm.

The structure or topology of the control is intentionally varied between the positive and negative feedback control modes so that the average response of the system is stable.

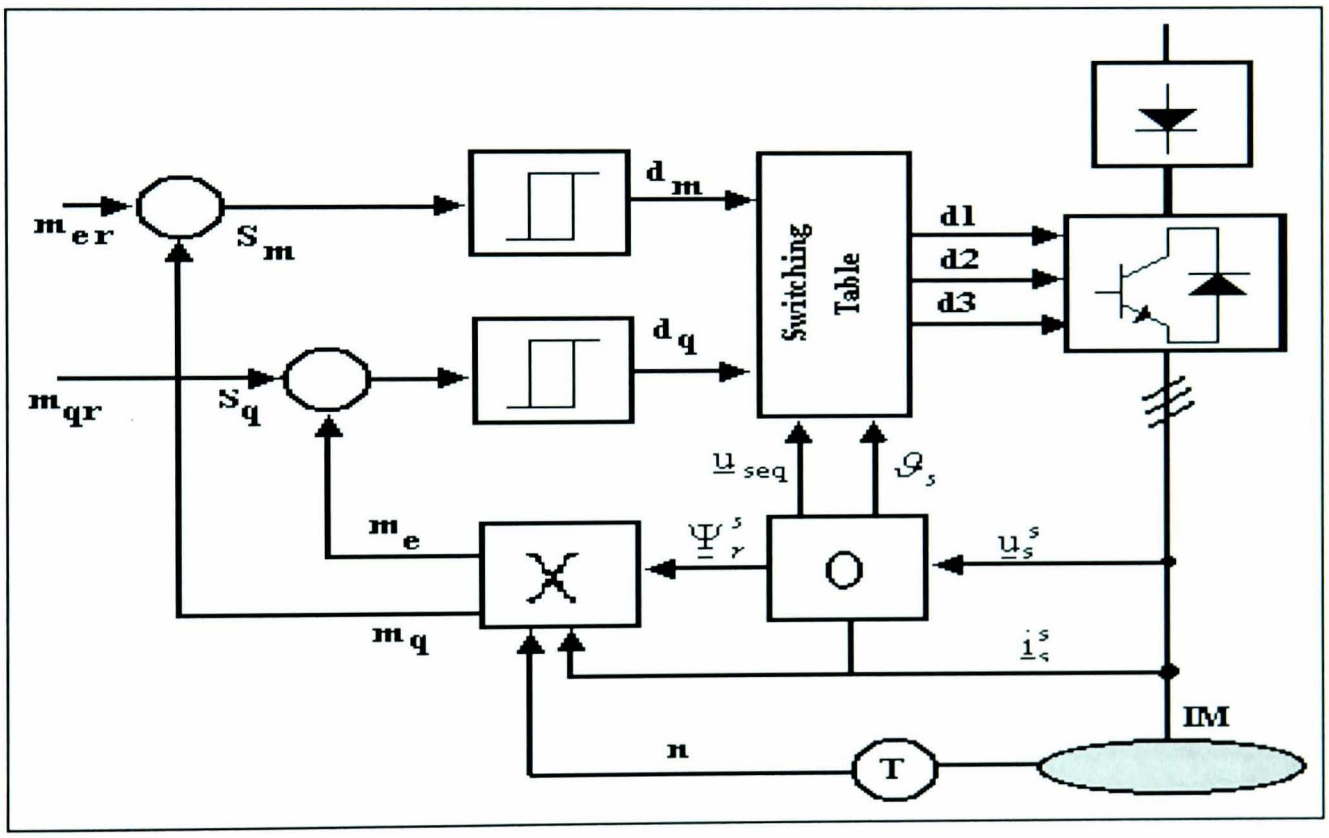


Figure 3.10 - *Block diagram of a sliding mode control system for the electromagnetic and the reactive torque*

With the help of a rotor flux observer '*O*' illustrated in Figure 3.10, the electromagnetic torque and the reactive torque can be computed. Inputs to this observer are the measured values of stator current and voltages. The torques m_e and m_q are compared to their references in order to form a switching law $\underline{s} = s_q + j\, s_m$. The controller must select binary signals *d1, d2, d3*, controlling the inverter branches in order to annul law $\underline{s}$.

The electromagnetic torque and the reactive torque are separately controlled by using two non linear controllers with hysteresis. The voltages $u_{s\alpha}$ and $u_{s\beta}$ can take the value u or $-u$ depending on the difference between reference and real torque values of m_{er} and m_e or m_{qr} and m_q respectively. Space vectors are used to establish the relation between voltages $u_{s\alpha}$, $u_{s\beta}$ and the terminal voltages of the stator *u1, u2* and *u3.*

The implementation of the variable structure control of an induction motor is presented by Arcker *et al* [32] and is illustrated in Figure 3.11.

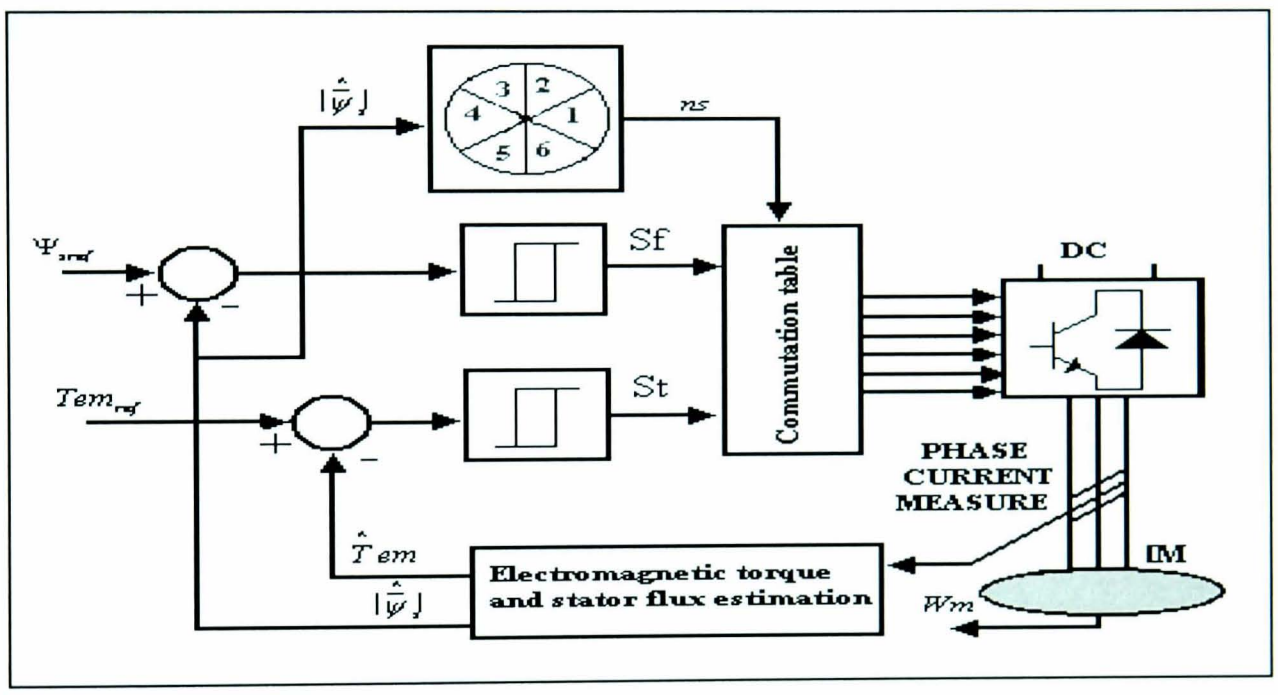


Figure 3.11 - *Variable structure control scheme*

The controller structure is based on :

⇒ Control of the stator flux

⇒ Control of the electromagnetic torque.

The main task of the controller is to estimate the stator flux, the electromagnetic torque, the stator flux position, the choice of the optimal voltage vector and the generation of the control signals of the inverter switches. The estimation of the stator flux and the electromagnetic torque is based on the following equations:

$$\frac{d\psi_{s\alpha}}{dt} = v_{s\alpha} - R_s I_{s\alpha} \qquad \textit{2-17}$$

$$\frac{d\psi_{s\beta}}{dt} = v_{s\beta} - R_s I_{s\beta} \qquad \textit{2-18}$$

Where $\bar{v}_{s\alpha\beta}$ is the stator voltage space vector imposed by the inverter, R_s the stator resistance and $\bar{I}_{s\alpha\beta}$ the stator current vector. By choosing the right voltage space vector at each switching instant, it is possible to make the stator flux following a circle reference.

In this case the error between the flux magnitude and its reference is maintained within the limits set by the hysteresis band $(2\Delta\psi)$. That is

$$\psi_{sref} - \Delta\psi < |\bar{\psi}_s| < \psi_{sref} + \Delta\psi \qquad \textit{2-19}$$

The electromagnetic torque T_{em} is represented by the following equation:

$$T_{em} = \frac{3}{2}(\psi_{s\alpha}.i_{s\beta} - \psi_{s\beta}.i_{s\alpha}) \qquad \textit{2-20}$$

The measurement of two phase currents are needed to estimate the control variables, $\hat{\psi}_s$ and $\hat{T}_{em.}$ and the estimated values are compared to their references ψ_{sref} and T_{emref}. The errors of the flux and torque magnitudes are used by the hyteresis controllers in order to determine Sf and St, which are used to choose the right space vector. The stator flux trajectory is divided to 6 zones according to:

$$(ns\text{-}1).\frac{\pi}{6} <= \leq \theta_s < ns.\frac{\pi}{6} \qquad \textit{2-21}$$

The stator flux position ϑ_s in αβ-frame can be calculated as follow:

$$\theta_s = \arctan(\frac{\psi_{s\beta}}{\psi_{s\alpha}}) \qquad \textit{2-22}$$

Thus the knowledge of the sector on which the stator flux involves is sufficient to allow the determination of the voltage vector to be applied. The VSC is designed to achieve very fast torque response.

The important features of the sliding mode controller which make it so attractive for application in variable speed drive are high accuracy, fast dynamic response, good stability, simplicity of design.

This chapter has reviewed the most recent research activities focused on the development of various induction motor control strategies. Particular attention was devoted to the work concerned with vector control applications to induction motor. It is important to note that in developing a control system, the design process is as important as the final product itself. The next chapter describes some of the electronic design tools used in the development process of the control system.

Chapter 4

Modern Control Systems Design Using CAD Techniques

4.1 Trends in Control System

The function of a control mechanism is to maintain certain essential properties of a system at a constant value under all conditions. Historically control systems which are simple but effective have been employed in water regulation and control of the liquid level in wine vessels for centuries. Some of these concepts are still used today, for example the float system in the water tank of the toilet flush. However, modern control systems used in today's industry are much more complex and owe their beginnings to the development of Control Theory. In 1868, Maxwell presented the first mathematical analysis of feedback control. It was during this period that systematic studies into control systems and feedback dynamics began. One significant development was the well known Routh's Stability Criterion (1877) which won E.J. Routh the Adam's Prize for that year.

The early 20th century saw the beginning of what is now known as *Classical Control Theory*. Minorsky's work (1922) on the determination of stability from differential equation describing the system (characteristic equation) and Nyquist's development (1932) of a graphical procedure for determining stability (frequency response) contributed immensely to the study of control theory.

In 1934, Hazen introduced the term 'servomechanism' to describe position control systems in his attempt to develop a generalised theory of servomechanism. Two years later, the development of the *Proportional-Integral-Derivative (PID) controller* was described by Callender et al. (1936). Control Theory, like many branches of engineering, underwent significant developments during World War II. Based on Nyquist's work, in (1945) H.W. Bode introduced a method for feedback amplifier design, now known as the *Bode plot*. In 1948, *root locus* method of design and stability analysis was developed by W.R. Evans. However following the introduction of digital computers in the 1960s the approach to system control began to change and the use of frequency response and characteristic equations began to give way to the solution of Ordinary Differential Equation (ODE), that describe system performance.

This led to the birth of *Modern Control Theory*. While the term Classical Control Theory is used to describe design methods of Bode, Nyquist, Minorsky and similar workers, Modern Control Theory relies on ODE design methods, like the *State Space Approach*, which are more suitable for computer aided engineering. Both these branches of control theory rely on mathematical representation of the control plant from which its performance is derived. To address the issues of non-linearities and time-variant parameters in plant models, control strategies that continuously adapt to the variations of plant characteristics have been introduced. Generally known as Adaptive Control Systems, they include techniques such as self-tuning control, H-infinity control, model referencing adaptive control and sliding mode control. Studies also include the use of non-linear state observers to continuously estimate the parameters of the control plant. They can be employed to tackle the issue of *non-observability*, that is the condition whereby not all of the required states are available for feedback. This may present a cheaper solution because it does not require as many sensors to be used and is adopted in applications such as variable speed drives, or where it is physically difficult or even impossible to obtain the feedback states such as in a nuclear reactor.

In many instances, the mathematical model of the plant is simply unknown or ill-defined, leading to greater complexities in the design of the control system. It has been proposed that *Intelligent Control Systems* give a better performance in such cases.

Unlike conventional control techniques, intelligent controllers are based on *Artificial Intelligence* (AI) rather than plant model. They imitate the human decision-making process and can often be implemented in complex systems with more success than conventional control techniques. AI can be classified into expert systems, fuzzy logic, artificial neural networks and genetic algorithms. With the exception of expert systems, these techniques are based on *soft-computing* methods. This means that they are capable of making approximations and 'intelligent guesses' where necessary, in order to establish a 'good enough' result under a given set of constraints. Intelligent control systems may employ one or more AI techniques in their design.

4.2 Traditional Control

Traditional control systems are normally implemented using analogue and/ or digital hardware. In its relatively short existence, digital computer technology has touched, and had a profound effect upon, many areas of life. Its enormous success is due largely to the flexibility and reliability that computer systems offer to potential users. This, coupled with the ability to handle and manipulate vast amounts of data quickly, efficiently and repeatedly, has made computers extremely useful in many varied applications. In control systems the digital computer can act directly as the controller or to provide the enabling technology that allows the design and implementation of the overall system, to obtain satisfactory performance.

Indeed, digital controllers have been used to give results as good, or better than analogue controllers in numerous cases, with the added feature that the control strategies can be varied by simply reprogramming the computer instead of having to change the hardware. In addition, analogue controllers are susceptible to ageing and drifting, which in turn causes performance degradation. These advantages have attracted many users to adopt digital technology in preference to conventional methods. With advances in VLSI (Very Large Scale Integration) and denser packing capabilities, faster integrated circuits can be manufactured which result in quicker and more powerful computers. Therefore, application to control areas which a few years ago were considered to be impractical or impossible because of computer limitations, are now entering the realms of possibility.

4.3 Microcomputer Control

The superiority of microcomputer control over conventional hardware based control can be recognized for complex drive control systems. The simplification of hardware saves control electronics cost and improves system reliability. Digital control inherently improves noise immunity, which is particularly important in this work where there are large power switching transients in the converters. In addition software control algorithms can easily be altered or improved in the future without changing the hardware. Another important feature is that the structure and parameters of the control system can be altered in real time, making the control adaptive to plant characteristics. The complex computation and decision taking capabilities of microcomputers enables the application of modern optimal and adaptive control theories to optimise the drive system performance. In addition, powerful diagnostics can be written in the software. Microcomputer technology has rapidly developed and today the use of an efficient high level language with large hardware integration and VLSI implementation of the controller is easily possible.

4.4 Digital Signal Processing Control (DSP)

DSP chips are general purpose data processors initially created for applications that handle large amounts of data such as data acquisition, image enhancement and processing, remote sensing, voice synthesis and recognition, telecommunications. The DSP architecture is adapted to handle mathematical problems in real-time. It implements functions such as Fast Fourier Transform (FFT), convolution, etc. The application of DSPs has now been extended to electric drive control because they extensively use many of the typical DSP function as part of the speed and torque control algorithms.

The first generation motion control designs used microcontrollers for position, speed, or current control and used resolver-to-digital converters (AD2S8x, AD2S9x) for rotor/shaft position feedback. These designs used analogue circuitry to close the current loop and could not meet the emerging requirements for more precise control and faster response time.

Second generation designs use DSPs for motor control environments, adding the DSP benefits of greater energy efficiency, more precise control, and less wear and tear of moving parts. These designs feature chipsets, which integrate the A/D converter, PWM control and vector transformation. Using fixed-point DSP chipsets, digital current loops are now more cost effective providing greater precision and faster response times
Analogue Devices uses an architecture that off-loads functionality from the DSP/Microcontroller. The onboard ADC reads analogue current input from sensors, generates PWM switching signals, and performs vector transformations. Its integrated design provides reduced costs, board space as well as a reduction in design and test time.

Third generation devices will move toward fully integrated solutions for motor control and will use a hardware description language (VHDL) as a unique EDA environment for the system modelling, evaluation and controller design, an approach pioneered in this thesis. The applications of DSP technology and the impact on induction motor control are reviewed.

Kim, *et al* [37] use an extended Kalman filter to estimate speed. The filter is employed to identify the speed and the rotor flux based on measured quantities of stator currents and d.c. link voltage. The proposed extended Kalman filter is implemented using a 32- bit floating point DSP, TMS320C30. In [38] the implementation of a field oriented control algorithm and a non linear observer using a system based on a TMS320C40 is proposed. Comparison with results given by a conventional flux estimator show excellent low sensitivity to rotor resistance variation.
Other aspects of induction motor drive implementations are reported in literature by Navrapescu, *et al* [39] and Moynihan, *et al* [40], for example, the discrete time induction machine mathematical model for DSP implementation is presented in [39].

The model was simulated and implemented using a Motorola signal processor DSP56002. The simulation and experimental results proved the accuracy of this model when tested under different conditions. These models can be adapted for micro controller applications. Present preoccupations are in the direction of creating the necessary base for an ASIC implementation which could offer better implementation performance.

4.5 Fuzzy Logic Control (FLC) and Artificial Neural Network (ANN)

In recent years, enormous research activities have been carried out in the field of Fuzzy Logic Control *(FLC)* and Artificial Neural Network *(ANN)* applied to various control and signal processing applications in power electronics and drives [41, 42, 43, 44, 45, 46].
The on line-tuning scheme for indirect field oriented induction machine drives is introduced by Zhen and Xu [41] in which two fuzzy slip compensators are designed to handle machine parameter deviations. One fuzzy compensator is used to on line tune T_r in the slip calculator when the machine is working in the speed tracking mode, while the other is used to compensate T_r when the machine is working in the constant speed mode. A good performance for an indirect field oriented induction machine drive is achieved in terms of overshoot, steady state error, torque disturbance and variable speed tracking.

In the last few years, the *ANN* has represented an emerging technology rooted in many disciplines, including identification and control systems [44, 45, 46, 47]. An *ANN* can be trained to emulate the unknown non linear plant dynamics by presenting a suitable set of input/output patterns generated by the plant. Neural estimation technique for induction machine flux presented in [44], compared the performances of an *ANN* with those of classical estimators. The result of the simulation work proved that computation calculation time is reduced compared to the classical estimator.
The work presented by Dinu [47] concentrated on developing a current control strategy that is suitable for neural network implementation. The ratio between the operation precision and the complexity of the hardware implementation is controlled by altering the number of neurones in the corresponding neural network.

The control strategy can be applied to a large range of three phase power systems including induction motors. One of the advantages of the approach is the reduction in the complexity of the control system achieved by eliminating the need for large external *EPROM* implemented look-up tables. Using this method increases the calculation speed of the controller, and enables the rescaling of the controller structure while exercising good control over the performance complexity ratio [48].

Presently, many digital control systems are microprocessor based, primarily because of the availability of control Integrated Circuits (ICs), cheaper memories and tremendous advancements in data handling capabilities. A big step forward in control is the use of Application Specific Integrated Circuits (ASICs), designed using modern Computer Aided Design (CAD) / Electronic Design Automation (EDA) techniques.

4.6 Electronic Design Automation (EDA)/CAD Techniques

Electronic Design Automation (EDA) enables this transition to take place with a higher degree of confidence than was previously possible. The use of (EDA) allows the digital design engineer to create, simulate and verify a design without the need to breadboard a prototype, allowing complex systems and ideas to be evaluated in a short period of time. In addition, an extremely high confidence in the final product before committing to experimental hardware [49] is possible. EDA tools are widely used in the field of electronic design, especially in Application Specific Integrated Circuits (ASICs). In order to successfully develop an ASIC, it is important to have a detailed knowledge of the potential advantages and the limitations of the ASIC technologies. Furthermore, it is essential to manage the design flow strategy in order to limit the number of design iterations and to complete the design in a short time. With the improved performance of modern Computer Aided Design (CAD) tools, ASICs can be developed within a relatively short time, at low cost level.

The result is that ASICs are currently used in all the IC market segments, ranging from simple circuits for gadgets and toys, to complex, highly accurate systems, for example in measurement equipment or medical applications.

The design of electrical systems has traditionally followed a design procedure similar to the one shown in Figure 4.1. With the advent of EDA software, a new design methodology has been developed to take advantage of simulation techniques.

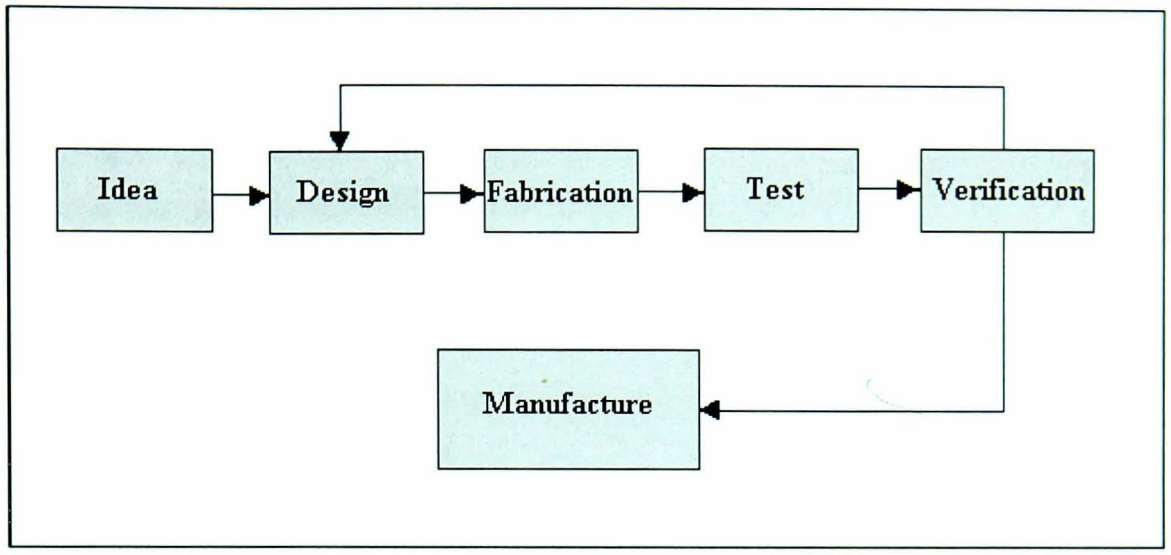


Figure 4.1 - *Traditional methods*

A comparison of the traditional with the simulation method using EDA illustrates the advantages of circuit and board level simulation in the production of a practical system. In the traditional design method Figure 4.1 the engineer begins with the idea , then normally proceeds to the circuit design stage on paper. The designer then continues through to the prototype stage using traditional construction methods. The prototype design is then tested and verified against its specification. At this point, if any conceptual fault is found a return to the design stage and, a repeat of the process would be required. This design cycle can be reduced considerably by removing three parts of the cycle before the design is verified. This technique is known as the simulation method and allows a product to be produced for the market in a much shorter time.

The simulation method illustrated in Figure 4.2 [50], allows the development of the design using the CAD system, whereby verification is carried out by simulating the circuit design using software models. At this point, design faults are identified and rectified without going through the costly path of prototype construction. The simulation method allows the design to be around 98% certain of working correctly first time [49].

The popularity of EDA tools rapidly increased with the widespread use of Application Specific Integrated Circuits (ASICs) and Field Programmable Gate Arrays (FPGAs) in the 1980s. In ASIC technology, the cost of correcting a design flaw late in the design process can be very high. The need for 'right-first-time' designs led to demands for reliable EDA tools. With increasing use of ASICs and FPGAs in power electronic control systems, EDA techniques are increasingly being employed. This has led to the development of a new design approach that relies more on verification by simulation, allowing new products to be developed and produced for the market in a shorter time.

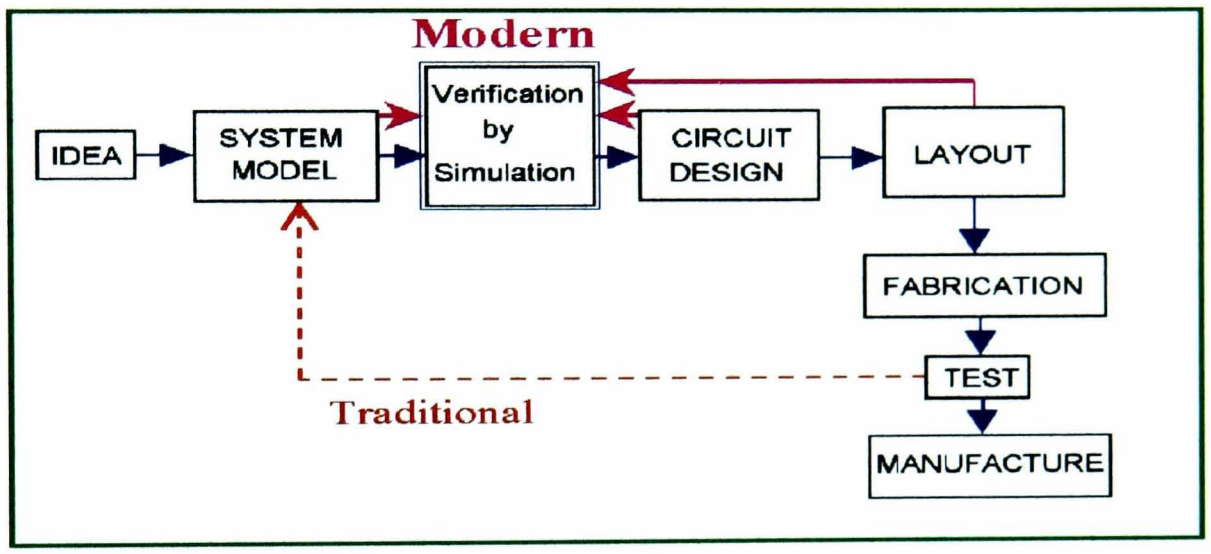


Figure 4.2 - *Simulation method*

4.6.1 Application Specific Integrated Circuits (ASIC's)

For many years the designers of electronic circuits and systems have been totally dependent upon the semiconductor manufacturers for the type of integrated circuit from which their circuits and systems may be built. In areas where very large volumes are required, such as calculators, televisions, radios and washing machines the semiconductor manufacturers have produced full custom designs.

The high cost of this process has prevented the exploitation of the size, speed, weight and reliability benefits of silicon design for all but the mass production market or certain military products.

The introduction of Computer Aided Design (CAD) in the 1980s has brought silicon design costs to a realistic level for an increased number of products. In most cases, if the total production of a few thousand pieces is anticipated, then it is likely that a semi-custom integrated circuit will prove viable. The uniqueness of a design in silicon is also an important commercial consideration. It will take a competitor much longer to copy the key

features of a silicon chip than it would for him to produce a comparable printed circuit board. Due to the availability of CAD systems, circuit and system designers now have the ability to produce the design to be implemented in silicon.
The stages in ASIC design are: schematic capture, simulation, logic optimisation and synthesis, placement and routing, layout versus schematic design rule check, and functions compiler. The design of a high-performance mixed-signal IC is inherently more difficult than the design of a logic IC. The variety of analogue and digital functions requires a cell-based approach. Thorough simulation and layout verification is necessary to ensure the functionality of the prototype ASIC. Re-design of large ASICs typically uses a high-level design language (HDL = Hardware Description Language) to help designers to document designs and to simulate large systems. The most common hardware description languages are Verilog and VHDL (the latter conforms to IEEE Standard 1076).

Programmable logic devices (PLDs) are uncommitted arrays of AND and OR logic gates that can be organized to perform dedicated functions by selectively making the interconnections between the gates. Recent PLDs have additional elements (output logic macro cell, clock, security fuse, tri-state output buffers and programmable output feedback) that make them more adaptable for digital implementations. The most popular PLDs are PALs (programmable array logic), PLAs (programmable logic array), and EPROMs. Programming of PLDs can be done by blowing fuses (in PALs) or by EEPROM or SRAM technologies which provide re-programmability. The main advantages of PLDs compared to FPGAs are the speed and ease of use without non-recurring engineering cost. The size of PLDs is, on the other hand, smaller than that of FPGAS. Current PLDs offer complexity equivalent to hundreds of thousands gates and speed in order of hundreds of MHz.

A disadvantage of ASICs in motion control systems is the lack of flexibility to modify or to adapt the design to different types of motor drives, once the chip is built. To change the design, even a small detail, it is necessary to go back to the initial design stages. The high development and fabrication cost for an ASIC can therefore be justified only in large volume production. In small-volume production and in prototyping stages, FPGAs offer a realistic alternative to full gate arrays design to implement specific motion control functions of high complexity requiring up to 1 million gates.

4.6.2 Field-Programmable Gate Array (FPGAs)

The Field-Programmable Gate Array (FPGA) is a user programmable device which plays an important role in the continuing evolution of Very-Large-Scale Integrated (VLSI) circuit technology towards denser and faster circuits. It already provides, for many applications, an adequate number of transistors in a single chip package for the functional blocks, routing network switches and memory. The prospects for further increase in both circuit density and speed of operation are excellent since VLSI circuit density continues to double every 2 to 3 years or so. The FPGA comprises thousands of logic gates, some of which are grouped together as configurable logic blocks (CLBs) to simplify higher level circuit design. The interconnections of the gates are defined by external SRAM or ROM. The advent of FPGA technology has enabled rapid prototyping of digital systems [51].

The FPGA has significant advantages for the development of prototype systems and their early introduction to the market. The benefits are similar to those associated with the introduction of the microprocessor in the late 1970s, such as programmability and adaptability, but with additional advantages in speed, compactness and design protection. The simplicity and programmability of the FPGA designate it as the most favourable choice for prototyping an ASIC. From the educational viewpoint, designing with FPGAs requires computer assistance at almost every stage of design, including detailed specification, simulation, placement and routing, and calls for an overall systematic design methodology [52].

4.6.2.1 FPGA Structure

Conceptually a re-programmable FPGA has a very simple structure as shown in Figure 4.3. FPGA families are differentiated by their chip-level architecture, and by the interchip wiring organisation. The simplest paradigm for FPGA use is the (silicon) "sand box", in which system structures can be built in the time it takes to write to a RAM, and with complexity levels defined by the chip architecture.

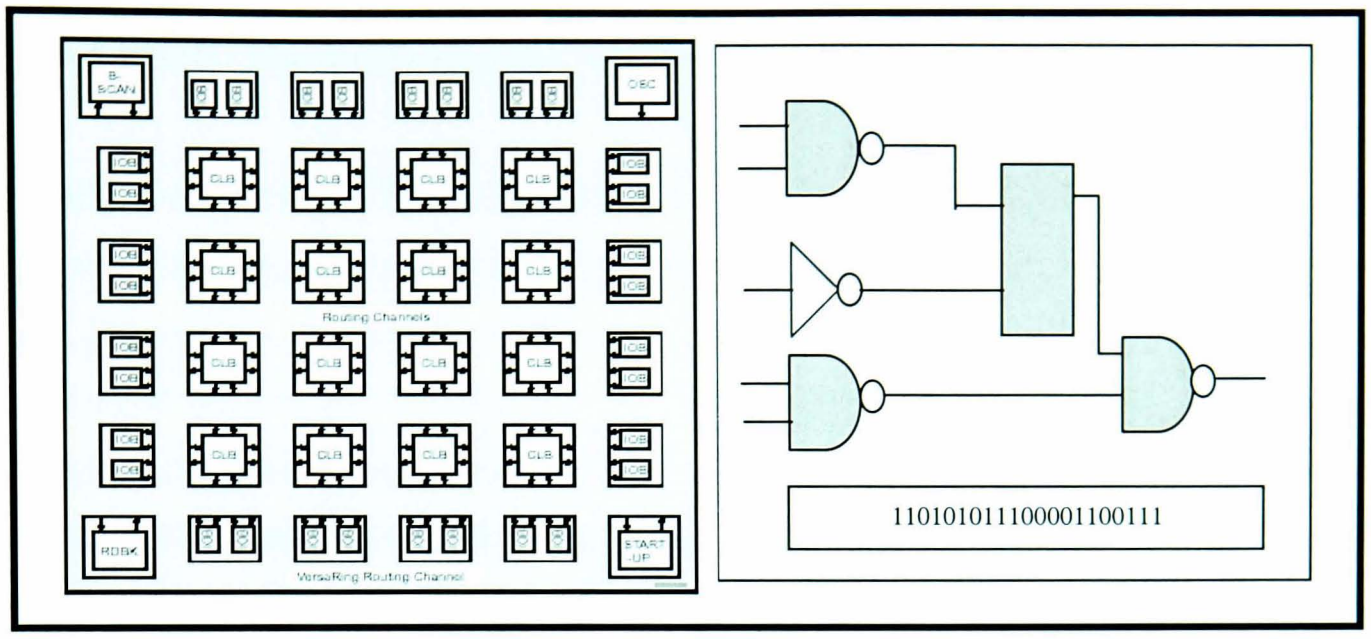


Figure 4.3 - *Conceptual structure of FPGAs*

FPGAs are a special class of ASICs which differ from mask-programmed gate arrays in that their programming is done by end-users at their site with no IC masking steps. An FPGA consists of an array of logic blocks that can be programmed and connected to achieve different designs. Current commercial FPGAs utilise logic blocks that are based on one of the following: transistor pairs, basic small gates (two-input NAND's and exclusive-OR's), multiplexers, look-up tables, and wide fan-in AND-OR structures.

FPGAs are implemented with regular flexible programmable architecture of Configurable Logic Blocks (CLBs) interconnected by a powerful hierarchy of versatile routing resources (routing channel) and surrounded by a perimeter of programmable Input/ Output Blocks as shown in Figure 4.3. They have generous routing resources to accommodate the most complex interconnected pattern. The devices are customised by loading configuration data into internal static memory cells. Reprogramming is possible an unlimited number of times. Current FPGAs offer complexity equivalent to one million gate conventional gate array and typical system clock speeds of hundreds of MHz. The size is much smaller than mask-programmed gate arrays but large enough to implement relatively complex functions on a single chip. The main advantage of FPGAs over mask-programmed ASICs is the fast turnaround that can significantly reduce design risk because a design error can be corrected quickly and inexpensively by reprogramming the FPGA. FPGAs are ideal for shortening design and development cycles, and also offer a cost effective solution for a higher production rate.

4.6.3 VHSIC Hardware Description Language (VHDL)

VHDL stands for the **V**HSIC **H**ardware **D**escription **L**anguage[53]. The VHSIC in turn refers to the **V**ery **H**igh **S**peed **I**ntegrated **C**ircuit program. This program was sponsored by the Department of Defence (DoD) of USA with the goals of developing high-speed integrated circuits and involved several major DoD contractors. During the course of this program the need for a standardised representation of digital systems became apparent. A team of DoD contractors was awarded the contract to develop the language and the first version was released in 1985. The language was subsequently transferred to the IEEE for standardisation after which representatives from industry, government and academia were involved in its further development. Subsequently the language was ratified in 1987 and became the IEEE 1076-1987 standard. The language was reballoted after five years and with addition of new features forms the 1076-1993 version of the language [53].

Ever since VHDL became an IEEE standard it has enjoyed steadily increasing adoption throughout the electronic system CAD community. In the last few years interoperability between models developed using VHDL environments from different CAD vendors has been improved by the establishment of the IEEE 1164 standard package and other standardisation efforts. Practically every major CAD vendor supports VHDL. The status of VHDL as an industry standard provides a number of practical benefits [53].

Conventional procedural programming languages such as C or Pascal typically describe procedures for computing a mathematical function or manipulating data, for example matrix multiplication or sorting. The execution of the program results in the computation of data values. Similarly, VHDL is a language for describing digital systems.

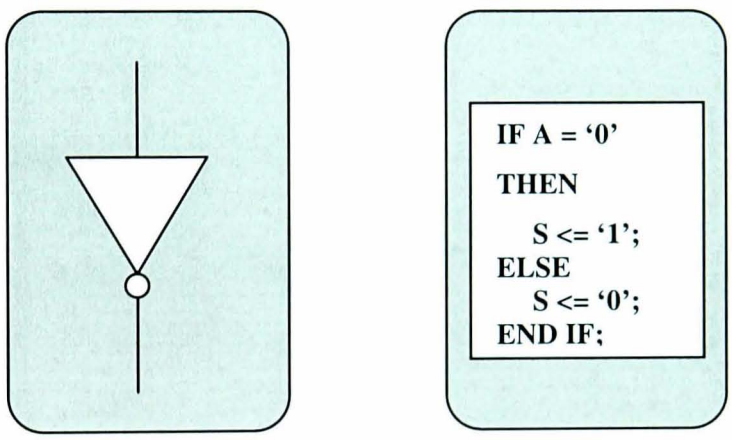
Investigations have been made for the use of VHDL in describing and simulating analogue systems. VHDL-AMS (Analogue Mixed Signal) is an extension of VHDL, which became a new standard on 1999. However, the language is still predominantly used in the design of digital systems.

4.6.3.1 Abstraction Levels and HDL's

Two domains can be identified to represent electronic systems:

- Structural domain.
- Behavioural domain.

Figure 4.4 identifies the elements of design description in each domain. A structural description could be a logic diagram, a behavioural description could be a set of logic equations and a physical description could be a standard cell layout for an ASIC. Figure 4.4 shows two different representations of an inverter, as example.



Structural domain Behavioural domain

Figure 4.4 - *Different domain representations of the same component*

4.6.3.2 Structural Design

Structural design simply connects elements together. It is common to most branches of engineering. Wires connect active and passive components in electronics, rivets and fasteners connect structural elements and beams in bridge building.

In both cases, the structural composition defines an architecture that imbues the system it defines with certain desirable performance characteristics, structural integrity, and so forth. A fuzzy, rather than precise objective in structural design, is to create an elegant

architecture that addresses cost and performance objectives in a simple but effective way. With FPGAs, structural design is achieved with a schematic drawing tool.

Hierarchical or structured design was rediscovered by hardware designers in the late 1970s, when the increasing complexity of full-custom design for products such as microprocessors created serious bottlenecks. Prior to this, software engineers had adapted traditional methods in building complex software products. Ideas such as modularization, information hiding, stepwise refinement were recast to help VLSI design tasks. In effect, programming techniques were applied to VLSI layout design because programming language constructs allowed crisp descriptions of artwork, particularly when it contained iterated or conditional features. Whole software design products, including silicon compilers and generators, evolved at this time. The complexity levels of present day FPGAs match the custom designs of that period, so that ASIC design methods developed then, remain highly relevant today.

4.6.3.3 Behavioural Design

Behavioural design, is creating a description of the functionality of the design functions. It may be embodied in a simulation model, a hardware description language program, a set of logic equations, or simply a truth table. Logic diagrams can become behavioural descriptions with the use of simple drawing conventions. Implementations are often developed by writing behavioural descriptions in a proprietary hardware description language [54].

4.6.3.4 Physical Design

Physical design involves producing sufficient data for a design to be manufactured, including testing. With FPGAs, physical design requires the definition of functions for the logic blocks and routing paths for the interconnections. This is analogous to producing a layout in VLSI design. The manufacturing data is held in a binary file for programming the part.

The main advantage of working at higher levels of abstraction is the reduction in the number of elements the designer has to consider. Moreover, high level descriptions in the behavioural domain imply the possibility of modelling the system behaviour, regardless of

the final structure, target technology and implementation details. VHDL makes possible the description of the digital systems at any abstraction level, using behavioural or structural information. The structural description is defined as an interconnection (netlist) of previously defined components stored in a library, while the behavioural description does not give information about the system structure, it only describes its behaviour. Consequently behavioural descriptions are closer to the functionality of the system, while the structural one considers implementation details. The complexity associated with behavioural description is much lower than the structural one.

The transformation of a high abstraction level description into another one at a lower abstraction level, is called **synthesis**. The transformation of a behavioural description into another one in the structural domain with the same abstraction level is sometimes called **synthesis**, although, if a particular library or technology is considered in this step, it can be called mapping. The transformations made within a domain at the same abstraction level to improve the system performances can be called **optimisation** or **refinement**.

4.6.4 Design Flow in Top Down Methodology

Top-down design was a major revolution in the techniques to engineer circuits. The jump from logic gates to abstract behavioral descriptions was similar to the jump from transistor-to logic-level (Boolean) design. It always takes awhile for a new design methodology to be proven and accepted. Brave designers, standardization, and government mandates were the catalysts for VHDL growth. Improved tools followed, and the field of logic synthesis fueled VHDL's expansion.

Design methodologies can be classified into two groups:

- Bottom-up - where the designer builds the system from elementary components, such as logic gates or transistors. This approach needs a previous phase in which the system is partitioned into smaller blocks.
- Top-down - where the designer builds the system from a functional specification through a synthesis process to the final implementation details Figure 4.6.

The first approach may be considered as the traditional design method, and is mainly based on schematic capture and simulation. The top-down approach is the method used in conjunction with VHDL, and it is based on code generation, simulation, and synthesis [55].

Top-down approach (Figure 4.5) is best suited for VHDL based design methodologies and their widespread use has been possible because of the use of CAD tools related to synthesis and simulation.

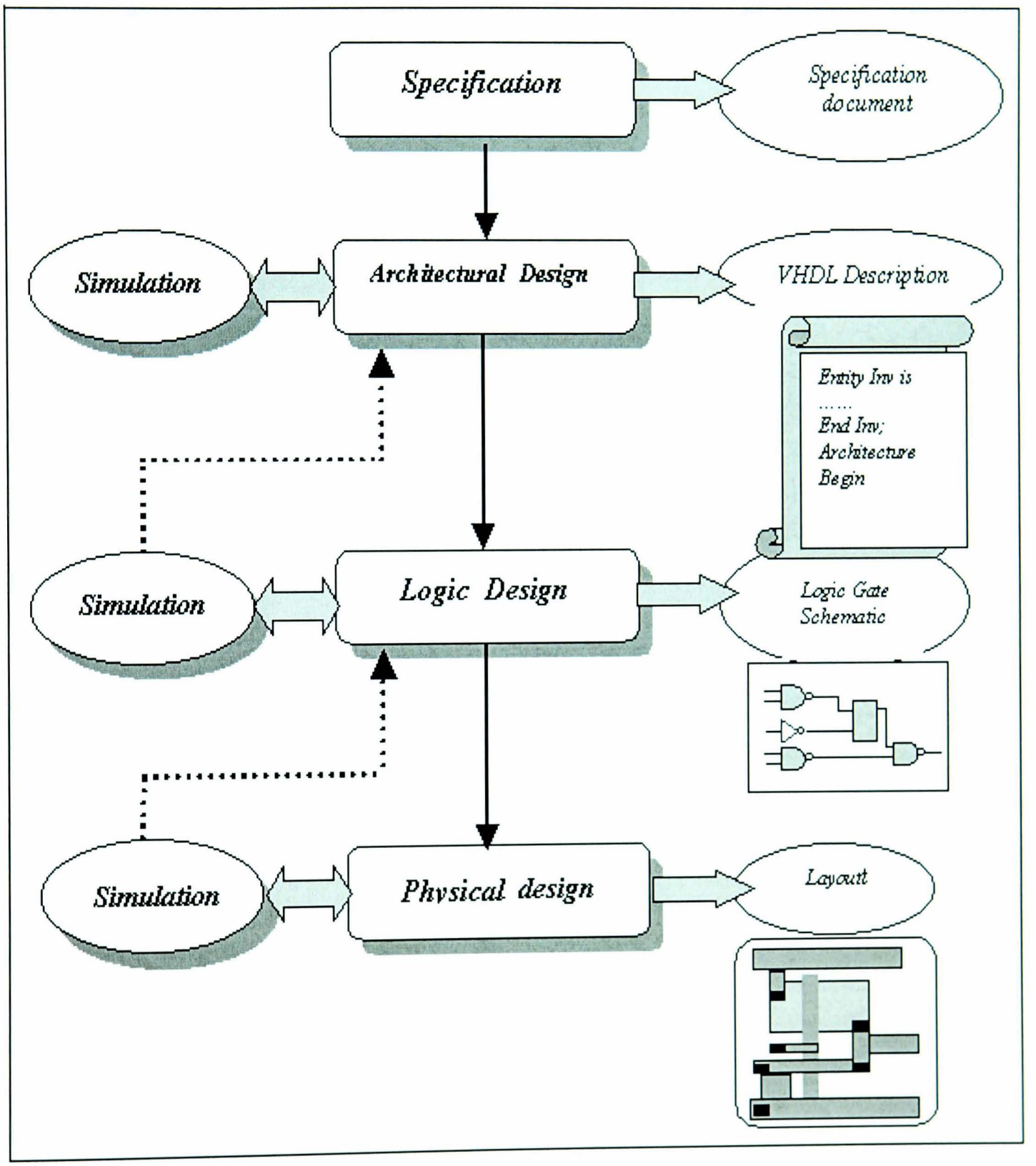


Figure 4.5 - *Top-down methodologies*

4.7 Xilinx Foundation Series

The Foundation Series is an EDA software by Xilinx Inc. for designing and implementing programmable hardware such as Field Programmable Gate Arrays (FPGAs) and Programmable Logic Devices (PLDs). The main component of the software is the Foundation Project Manager, an application that manages the EDA tools in the software and maintains a unified environment for the user. It comprises five groups: Design Entry, Simulation, Implementation, Verification and Programming Figure 4.. There are three Design Entries: HDL Editor, FSM (Finite State Machine) Editor and Schematic Editor. They allow the project design to be described either as a HDL program, a state machine description or as a schematic design. After the Design Entry stage, the design can be *synthesised*, a process that converts the design, whether it is a HDL program or a schematic, into a netlist format. The netlists contains the structural description of the design and are used for functional simulation. At this stage, it is not yet specific to any technology.

In order to download the design into hardware, the target technology has to be specified. The netlist is complied into a format that is compatible to the targeted device in a process that is called *implementation*. This is followed by accurate timing simulation. It is important to note that the targeted device has to be confirmed at the start of the implementation procedure. The Xilinx S40PQ208 FPGA device is used for the implementation of the final project. It is sufficient to point out that the final product of this procedure is a bitstream file, which can be directly downloaded into the targeted device via the serial or parallel interfaces of a PC [56].

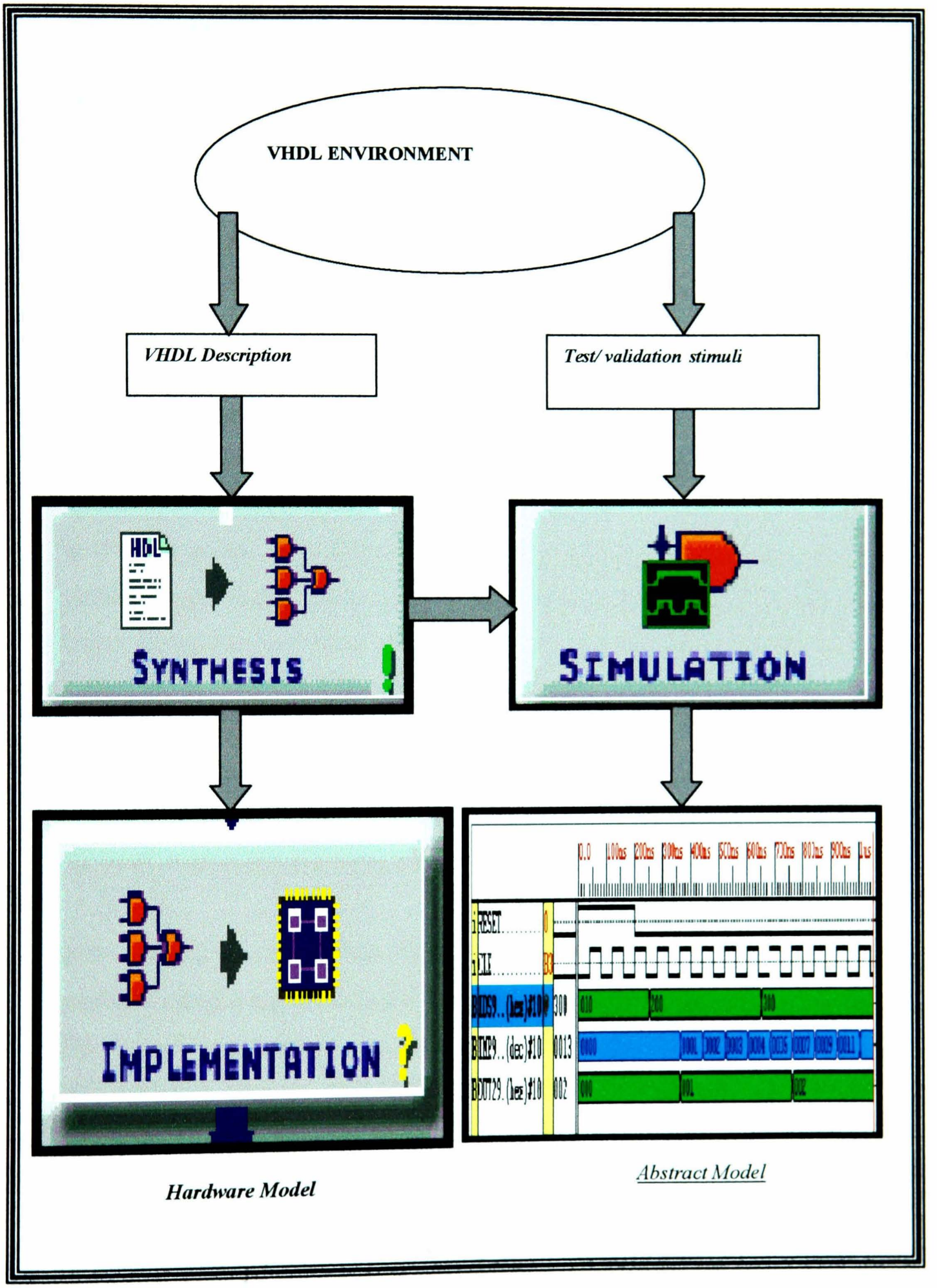


Figure 4.6 - *A common environment for abstract and hardware*

4.8 The Advantages of VHDL

♦⇒ VHDL provides several key advantages. Possibly the most important is the ability to perform top-down design. Top-down design first describes a system at a very high level of abstraction, like a specification. Designers simulate and debug the system at this very high level before refining it into smaller components. The method describes each smaller component at a high level and debugs it alone and with the other components in the system. The design continues to be refined and debugged until it is complete down to its lowest building blocks. Mixed-level design occurs when some components are at a more detailed level of description than others, but all components are simulated and debugged together. VHDL provide a way of describing and simulating the system and its components.

♦⇒ The power of top-down design is that engineers can discover and correct system problems early in the design cycle. They can design and fine-tune component interfaces before completing the actual components. They can simulate components as part of the system before making a silicon commitment. Experts estimate that 90% of all ASICs work right the first time on their own, but only 50% of them work right the first time in the system. Top-down design with system simulation can improve the success rate of the system. Further, software designers can develop system software without waiting for hardware availability. The bottom line is lower cost and faster time to market [57].

♦⇒ During the years that designers were creating VHDL, digital designers were manually refining top-down design procedures from high-level descriptions to logic gates. The field of logic synthesis was still experimental. Logic synthesis describes a circuit at a high level in a language such as VHDL and then uses a software tool that automatically generates the logic-gate implementation for the design. This advanced technology can significantly speed design because engineers spend less time producing logic-gate representations and can concentrate on overall design issues, such as system requirements and timing. Synthesized designs can be smaller and faster than manually created ones, and smaller designs usually mean lower cost. Synthesized designs also can cut debug time

because the designs are generally "correct by construction." As logic synthesis emerged as a mainstream technology, VHDL fit in perfectly as a language to begin synthesis.

♦⇒ Another important advantage that VHDL brought is standardization. It enables manufacturers to document all electronic systems and components in a common format, which allows various interested parties to understand and participate in a system's development. VHDL also provides the ability to capture designs in a standard format for archiving.

This chapter presented a review of Modern Control System Design using CAD Techniques, including a brief introduction to the concept of Fuzzy Logic Control and Artificial Neural Network. In addition, this chapter has introduced the VHDL design language and its main advantages. These make the software suitable to be used for the analysis, development and design of the novel control system, as presented in the following sections. The work described in the remainder of the thesis aims to utilize the precepts and present a new holistic approach to design.

Chapter 5

System Modelling and Simulation

A new approach to the modelling, simulation and controller design of a complete induction motor electric drive system is described in this chapter. The novel technique uses a hardware description language (VHDL) as a unique EDA environment for the system modelling, evaluation and controller design.

Modelling and simulation are inseparable procedures that include the complex activities associated with the construction of models representing real processes, and experimentation to obtain data on the behaviour of the system being modelled. Modelling deals primarily with the relationships between actual dynamic processes and appropriate models while simulation refers to the relationships between the model and the simulation tool. To complete the exercise the model time responses, being the outputs of the simulation tool, are evaluated in connection with the process being studied. Mathematical models of dynamic systems and computer simulation find application in technical and non-technical areas. The purpose of studying systems through modelling and simulation is to produce performance without actually constructing or operating real processes. The benefits of using system modelling and simulation may be summarised as:

- ♦ increase understanding of mechanisms in the studied process;
- ♦ prediction system behaviour in different situations.
- ♦ enabling the design and evaluation of synthesized control systems;
- ♦ estimation of those process variables which are not directly measurable;
- ♦ testing the sensitivity of system parameters;
- ♦ optimizing system behaviour;

Broadly speaking, simulation is defined to be an experiment with logical and mathematical models, especially of the dynamic kind, that are characterized by a mix of differential and algebraic equations.

5.1 The Space Vector Model of the a.c. Machine.

The space vector quantities (voltages, currents, m.m.f.s, flux densities, flux linkages, etc) are introduced by using both mathematical and physical considerations. For simplicity, a smooth air gap a.c. machine is considered with symmetrical two-pole and three-phase windings. Figure 5.1 shows the cross section of the machine under consideration .

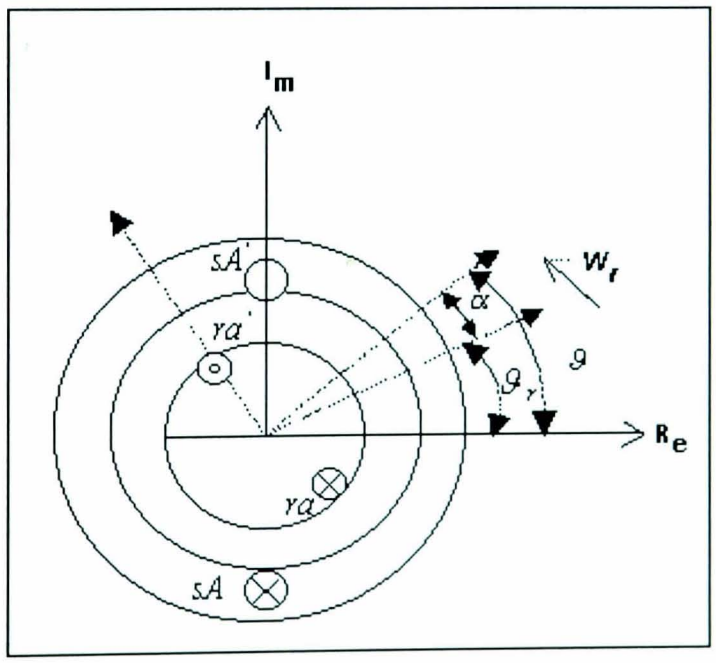


Figure 5.1 - *Cross section of an elementary symmetric 3-phases machine*

The stator and rotor windings are shown as single, multiple-turn full pitch coils situated on the two sides of the air-gap. These represent distributed windings, which at every instant produce sinusoidal m.m.f. waves, centred on the magnetic axes of the respective phases. The phase windings are displaced by 120 electrical degrees from each other. ϑ_r is the rotor angle which is the angle between the magnetic axes of stator winding sA and rotor winding *ra*. In general, the speed of the rotor is $\omega_r = \frac{d\vartheta_r}{dt}$ [58].

5.1.1 The 3-Phase (abc) to 2-Phase ($\alpha\beta$) Transformation

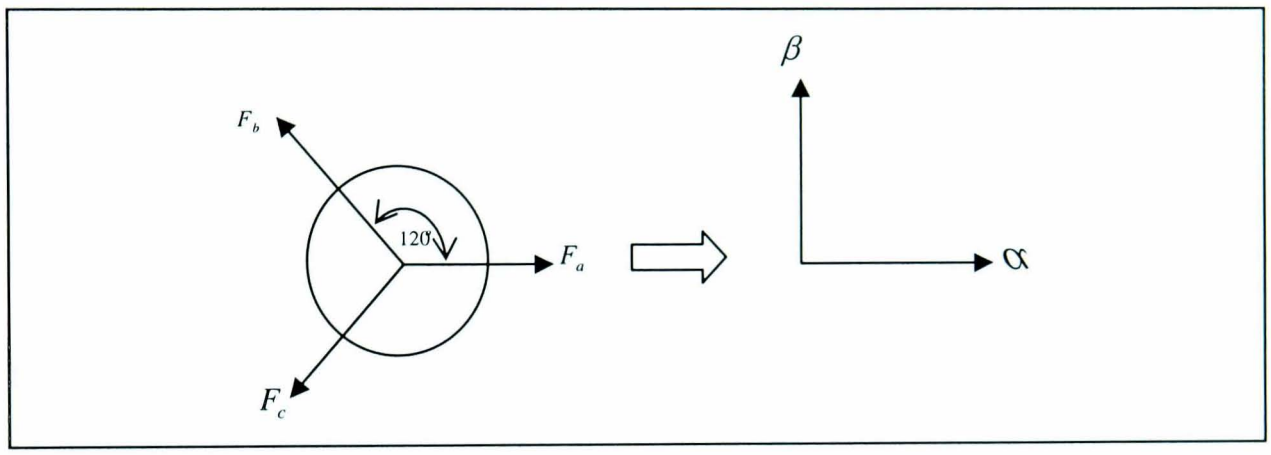


Figure 5.2 - *3-phase to 2-phase transformation*

The axis of phase α is chosen to coincide with phase a axis Figure 5.2. The 3-phase m.m.f.s can be resolved with respect to the two-phase axes, where F quantities are voltages, currents or fluxes.

$$F_\alpha = F_a + F_b\cos(120) + F_c\cos(-120)$$
$$F_\beta = F_a\cos(90) + F_b\sin(120) + F_c\sin(-120) \qquad \textbf{5-1}$$

which is equal to

$$F_\alpha = F_a - \frac{1}{2}F_b - \frac{1}{2}F_c$$
$$F_\beta = 0 + \frac{\sqrt{3}}{2}F_b - \frac{\sqrt{3}}{2}F_c \qquad \textbf{5-2}$$

In matrix form:

$$\begin{bmatrix} F_\alpha \\ F_\beta \end{bmatrix} = \begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \end{bmatrix} = \begin{bmatrix} F_a \\ F_b \\ F_c \end{bmatrix} \qquad \textbf{5-3}$$

The transformation matrix of equation **5-3** is a [2×3] matrix therefore to obtain an inverse matrix the next step is to create a [3×3] matrix by including a third equation describing the 3-phase system.

$$F_\gamma = m(F_a + F_b + F_c) \qquad \textbf{5-4}$$

Where m = a scalar parameter.

$F_\gamma = 0$ under balanced 3-phase conditions.

Therefore equation **5-3** becomes:

$$\begin{bmatrix} F_\alpha \\ F_\beta \\ F_\gamma \end{bmatrix} = \begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \\ m & m & m \end{bmatrix} = \begin{bmatrix} F_a \\ F_b \\ F_c \end{bmatrix} \qquad \textbf{5-5}$$

If α, β are assumed to have N_2 turns and a,b,c are assumed to have N_1 turns then:

$F_\alpha = N_2 i_\alpha$ and $F_\beta = N_2 i_\beta$ and $F_\gamma = N_2 i\gamma$ and.

$F_a = N_1 i_a$ and $F_b = N_1 i_b$ and $F_c = N_1 i_c$. **5-6**

Therefore matrix **5-5** can be written as:

$$\begin{bmatrix} N_2 i_\alpha \\ N_2 i_\beta \\ N_2 i_\gamma \end{bmatrix} = \begin{bmatrix} 1 & -\frac{1}{2} & \frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \\ m & m & m \end{bmatrix} \begin{bmatrix} N_1 i_a \\ N_1 i_b \\ N_1 i_c \end{bmatrix} \qquad \textbf{5-7}$$

Equation **5-7** can be written as:

$$\begin{bmatrix} i_\alpha \\ i_\beta \\ i_\gamma \end{bmatrix} = \frac{N_1}{N_2} \begin{bmatrix} 1 & -\frac{1}{2} & \frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \\ m & m & m \end{bmatrix} \begin{bmatrix} i_a \\ i_b \\ i_c \end{bmatrix} \qquad \textbf{5-8}$$

$$[A] = \frac{N_1}{N_2} \begin{bmatrix} 1 & -\frac{1}{2} & \frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \\ m & m & m \end{bmatrix} \qquad \textbf{5-9}$$

For orthogonality [A] must satisfy the following matrix relation:

$$[A][A]^t = [1] \qquad \textbf{5-10}$$

$$\left(\frac{N_1}{N_2}\right)^2 \cdot \begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \\ m & m & m \end{bmatrix} \begin{bmatrix} 1 & 0 & m \\ -\frac{1}{2} & \frac{\sqrt{3}}{2} & m \\ -\frac{1}{2} & -\frac{\sqrt{3}}{2} & m \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \textbf{5-11}$$

Therefore

$$\left(\frac{N_1}{N_2}\right)^2 (1\times(1+(-\frac{1}{2})(-\frac{1}{2})+(-\frac{1}{2})(-\frac{1}{2})=1$$

$$\left(\frac{N_1}{N_2}\right)^2 (1+\frac{1}{4}+\frac{1}{4})=1$$

$$\frac{N_1}{N_2}=\sqrt{\frac{2}{3}}$$

and

$$\left(\frac{N_1}{N_2}\right)^2 (m^2+m^2+m^2)=1$$

$$3*\left(\sqrt{\frac{2}{3}}\right)^2 m^2=1$$

therefore

$$m=\sqrt{\frac{1}{2}}$$

Hence, from equation **5-9**, matrix [A] can be written as:

$$[A]=\sqrt{\frac{2}{3}}\begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} \qquad \textbf{5-12}$$

Substituting into equation **5-8** yields the orthogonal transformation matrix:

$$\begin{bmatrix} F_\alpha \\ F_\beta \\ F_\gamma \end{bmatrix}=[A]\begin{bmatrix} F_a \\ F_b \\ F_c \end{bmatrix} \qquad \textbf{5-13}$$

The transformation relationship can also be written as:

$$[F_{\alpha\beta\gamma}] = [A][F_{abc}]$$
$$[F_{abc}] = [A]^t [F_{\alpha\beta\gamma}] \qquad \textbf{5-14}$$

Equation **5-14** represents the transformation relationships.

5.1.2 Commutator Transformation.

For a rotating coil system, a second transformation needs to be applied to fix the rotating 2-phase m.m.f vectors to a stationary frame. As with the stator parameters, the 3-to-2 phase transformation is carried out but in the case of the rotor, the equivalent ($\alpha\beta$) axes rotate at an angular speed ω_r.

The commutator transformation transforms the rotor quantities from a two axis rotating space-frame to a two-axis fixed space frame. The effect of applying this transformation is to replace the rotating winding m.m.f distribution with a stationary winding m.m.f distribution. This is presented below.

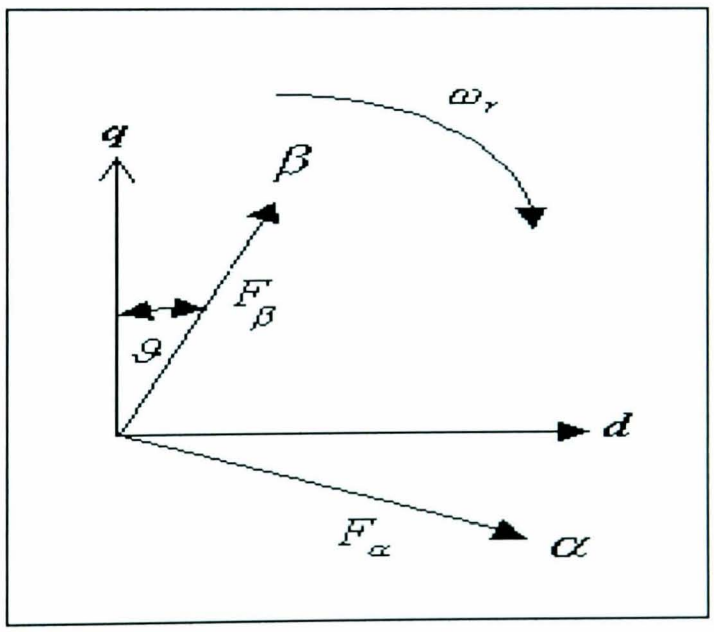


Figure 5.3- *The orthogonal DQ-axis*

From the rotating two phase coil system in, the following equations are obtained:

$$F_d = F_\alpha \cos\vartheta + F_\beta \sin\vartheta \qquad \textbf{5-15}$$

$$F_q = -F_\alpha \sin\vartheta + F_\beta \cos\vartheta \qquad \textbf{5-16}$$

$$F_\gamma = F_\gamma \qquad \textbf{5-17}$$

Note that the quantities of equation **5-17** in the two phase system have no physical significance with respect to an actual machine coil representation.

Equations **5-15** - **5-17** can be written in matrix form as:

$$\begin{bmatrix} F_d \\ F_q \\ F_\gamma \end{bmatrix} = \begin{bmatrix} \cos\vartheta & \sin\vartheta & 0 \\ -\sin\vartheta & \cos\vartheta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} F_\alpha \\ F_\beta \\ F_\gamma \end{bmatrix} \qquad \textbf{5-18}$$

$$\text{Let } B = \begin{bmatrix} \cos\vartheta & \sin\vartheta & 0 \\ -\sin\vartheta & \cos\vartheta & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \textbf{5-19}$$

Therefore the transformation relationship can be written as:

$$[F_{dq\gamma}] = [B][F_{\alpha\beta\gamma}] \qquad \textbf{5-20}$$

The corresponding inverse transformation may be expressed as:

$$[F_{\alpha\beta\gamma}] = [B]^t [F_{dq\gamma}] \qquad \textbf{5-21}$$

5.1.3 Transformation of a Rotating 3-Phase (abc) Quantity to Stationary 2-Phase (dq) Quantity.

From the relationships shown in previous sections it can be concluded that in order to transform a rotating 3-phase coil distribution to a stationary 2-phase coil distribution the application of matrices [A] and [B] given by relations **5-12** and **5-19** is required. This results in:

$$[F_{dq\gamma}] = [B][A][F_{abc}] \qquad \textbf{5-22}$$

Using equation **5-22**, the complete transformation is expressed as:

$$\begin{bmatrix} F_d \\ F_q \\ F_\gamma \end{bmatrix} = \sqrt{\frac{2}{3}} \begin{bmatrix} \cos\vartheta & \cos(\vartheta-120) & \cos(\vartheta+120) \\ -\sin\vartheta & -\sin(\vartheta-120) & -\sin(\vartheta+120) \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} F_a \\ F_b \\ F_c \end{bmatrix} \qquad \textbf{5-23}$$

$$\text{let } K = \sqrt{\frac{2}{3}} \begin{bmatrix} \cos\vartheta & \cos(\vartheta-120) & \cos(\vartheta+120) \\ -\sin\vartheta & -\sin(\vartheta-120) & -\sin(\vartheta+120) \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} \qquad \textbf{5-24}$$

Finally the transformation relationships between the (*abc*) axis and the (*dq*) axis can be expressed as:

$$\begin{aligned} [v_{dq\gamma}] &= [K][v_{abc}] \\ [i_{dq\gamma}] &= [K][i_{abc}] \\ [\psi_{dq\gamma}] &= [K][\psi_{abc}] \end{aligned} \qquad \textbf{5-25}$$

The transformation of both the stator and the rotor parameters expressed are summarized as:

- The transformation from (*abc*) to ($\alpha\beta$) is achieved by equation **5-13**.
- The transformation from ($\alpha\beta\gamma$) to ($dq\gamma$) is achieved by equation **5-18**.
- The transformation from (abc) to ($dq\gamma$) is achieved by equation **5-23**.

Finally, transformation relationships for stator quantities are given by:

$$\begin{aligned} [i_{dq}]_s &= [A][i_{abc}]_s \\ [v_{dq}]_s &= [A][v_{abc}]_s \end{aligned} \qquad \textbf{5-26}$$

and for the rotor parameters:

$$\begin{aligned} [i_{dq}]_r &= [C][i_{abc}]_r \\ [v_{dq}]_r &= [C][v_{abc}]_r \end{aligned} \qquad \textbf{5-27}$$

5.2 Modelling and Simulation of the Induction Motor.

Several control algorithms were developed in the late 1960's, to achieve torque control of induction motors. However, these methods met with only limited success, since they required too many complicated calculations involving many unknown model parameters and numerous approximations. With advances of faster switching and more easily controlled power electronic components, such as the Gate Turn Off (GTO) thyristor, the Bipolar Junction Transistor (BJT), the IGBT's, etc, researchers realised that new control schemes could be implemented for a.c. machines. In the 1970's a new algorithm for a.c. induction motor control, known as field oriented control was introduced that applied the two-vector method to the induction motor by separating the stator current into a flux-producing component and an orthogonal torque-producing component. The result is that, field orientation provides the same decoupled control of torque and flux, which is inherently possible in the d.c. machine.

5.2.1 Induction Machine Model.

The continuous-time electromechanical model of an induction machine is of fifth order and non-linear [59] and can be represented as:

5.2.1.1 Electrical Model

$$\frac{di_{ds}}{dt} = (R_s L_r i_{ds} + \omega_r L_m^2 i_{qs} - R_r L_m i_{dr} - \omega_r L_r L_m i_{qr} - L_r v_{ds}) / a_0$$

$$\frac{di_{qs}}{dt} = (\omega_r L_m^2 i_{ds} + R_s L_r i_{qs} + \omega_r L_r L_m i_{dr} - R_r L_m i_{qr} - L_r v_{qs}) / a_0$$

$$\frac{di_{dr}}{dt} = -(R_s L_m i_{ds} - \omega_r L_m L_s i_{qs} - R_r L_s i_{dr} - \omega_r L_r L_s i_{qr} - L_m v_{ds}) / a_0 \qquad \textbf{5-28}$$

$$\frac{di_{qr}}{dt} = -(\omega_r L_m L_s i_{ds} + R_s L_m i_{qs} + \omega_r L_r L_s i_{dr} - R_r L_s i_{qr} - L_m v_{qs}) / a_0$$

$$T_e = \frac{3}{2} L_m (i_{qs} i_{dr} - i_{ds} i_{qr})$$

where $a_0 = (L_m * L_m - L_r * L_s)$;

Where $i_{qs}, i_{ds}, i_{qr}, i_{qr}$ are the d-q axis stator and rotor current in the stationary reference frame, v_{ds}, v_{qs} are the stationary frame d and q axis stator voltages and ω_r is the rotor angular speed.

5.2.1.2 Mechanical Model

To define the behaviour of the induction motor, the system motion equation needs to be integrated with the *dq-axis* equations. The load torque of the motor may be due to friction, windage, acceleration and mechanical work. Neglecting the coulomb and static friction components, the torque due to friction is approximated to $T_F = B\omega_r$, viscous friction, where B is a constant for the system and ω_r is the rotor speed. The windage torque is combined with the viscous torque and is given by $T_{F+W} = D\omega_r$ where *D* is the damping constant. The torque component required to accelerate the system is expressed as:

$T_J = J\frac{d\omega_r}{dt}$, where J is the rotational inertia of the system in $Kg.m^2$.

The load torque T_l,, is the mechanical work required and can be expressed as a function of ω_r. Therefore, the developed electromagnetic torque T is given by:

$$T_e = J\frac{d\omega_r}{dt} + D\omega_r + T_l \qquad \textbf{5-29}$$

$$\omega_r = \frac{1}{J}\int(T_e - T_l)dt. \qquad \textbf{5-30}$$

where T_e = the electric developed torque.

5.2.1.3 Vector Control Modelling.

To implement indirect vector control the measurements required are the induction machine stator currents and the rotor speed or position. A rotor slip calculation is used to find the slip speed which is then integrated to give the slip value. Adding this to the rotor position measurement gives the rotor flux position and hence, the unit vectors required to perform the transformation between the stationary frame and the rotating frame quantities.

In order to achieve the performance required by servo applications, induction motor control is achieved using the vector control strategy. This allows high performance control of torque, speed or position to be achieved from an induction machine [59].
This method can provide at least the same performance from an inverter-driven induction machine as it is obtainable from a separately excited d.c. machine. Vector control provides decoupled control of the rotor flux magnitude and the current component generating the torque. The fast torque response is achieved by estimating, measuring or calculating the magnitude and position of the rotor flux in the machine.

5.2.1.4 Induction Machine Complete Model.

A software model for the simulation of the vector controlled induction machine system was developed using VHDL programming language introduced by Navabi [60]. This contains a mechanical block, an electric module and the vector control model block as illustrated in Figure 5.4. The simulation of the induction motor drive is based on the mathematical model equations **5-28** – **5-30**. This approach requires the development of numerical integration routines for solving the resultant differential equations that describe the system based on Euler's method.

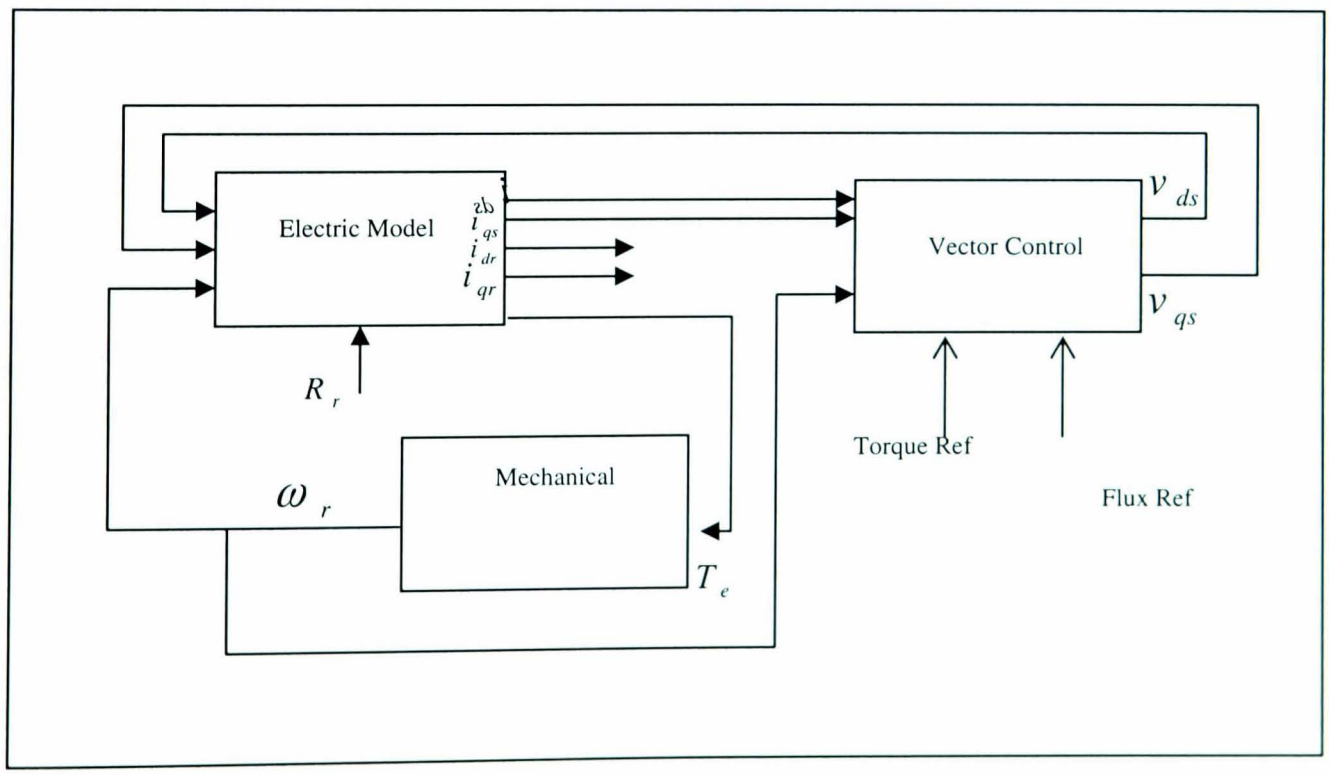


Figure 5.4 - *Vector controlled induction machine*

The equations of the mechanical model of the induction machine are **5-29** and **5-30**. A typical block diagram representing the set of differential equations is shown in Figure 5.4.

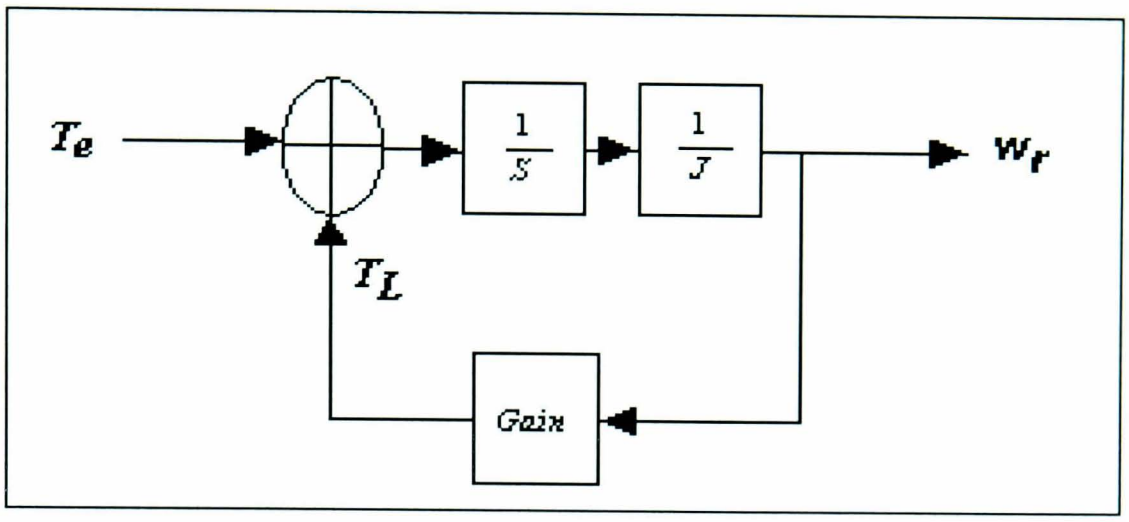


Figure 5.4 - *Block diagram of induction motor mechanical model*

5.3 VHDL Functional Simulation.

Although VHDL is a hardware description language and, as such, is used primarily for circuit design, it has the basic properties of any software programming language. It is therefore capable of implementing mathematical models. The VHDL code can hence be designed to represent the model of the vector controlled induction machine system. The complete source code that achieves this can be found in appendix ***A***. Since the system model is designed for the purpose of functional simulations, it is possible to declare the signals and variables as type ***REAL***. This gives the design a greater freedom in numerical analysis. It is also possible to incorporate complex mathematical functions, such as trigonometry, in the design.

The VHDL model for the vector controlled induction motor is configured using the entity ***Motor***. The input signals of the model are the voltages in the ***d-q axis*** and the load torque $\mathbf{T_L}$. Three output signals provide simulated information on the current in ***d-q axis*** and the actual rotor speed.

The data regarding the motor operation during simulation is stored in an output ASCII file (motor.txt). The file contains numerical data in matrix format. Each line in the matrix contains the set of quantities that characterise the motor operation at a certain moment in time: currents, voltages, speed and torque. The VHDL code for the system model is shown below:

```
LIBRARY math;
USE math.mathtyx.all;
USE std.textio.all;
-- Electrical+mechanical model
ENTITY motor IS
  PORT (vds,vqs,Tl: IN Real;
        ids,iqs,wr: OUT REAL);
END motor;
ARCHITECTURE arch_motor OF motor IS
  CONSTANT Rs: REAL := 5.9;
  CONSTANT Rr: REAL := 4.62;
  CONSTANT ls: REAL := 0.831;
  CONSTANT lr: REAL := 0.833;
  CONSTANT lm: REAL := 0.809;
  CONSTANT jr: REAL := 0.001;
  CONSTANT deltat:TIME:= 1000ns;
  CONSTANT dt: REAL :=1.0e-6;
  CONSTANT wc: REAL := 50.0;
  CONSTANT p: REAL := 4.0;
constant flag:integer:=1;
signal next_step: INTEGER := 1;
signal vdsr,vqsr: REAL :=0.0;
signal vdsr1,vqsr1: REAL :=0.0;
signal tt:real:=1.0;
FILE outf: TEXT IS OUT "C:\motor.txt";
BEGIN
PROCESS(next_step)
VARIABLE my_line: LINE;
VARIABLE a,ids1,idss,iqs1,iqss: REAL:=0.0;
VARIABLE idr1,idrr,iqr1,iqrr: REAL:=0.0;
VARIABLE Te,wr1,wrr: REAL :=0.0;
variable thetar,tr:real:=0.0;
variable t:real:=0.0;
CONSTANT d_space: STRING :="  ";
BEGIN
    IF next_step=1 THEN
      WRITE(my_line,wc);
      WRITE (my_line,d_space);
      WRITE(my_line,wrr);
```

```
        WRITE (my_line,d_space);
        WRITE(my_line,tt);
        WRITE (my_line,d_space);
        WRITE(my_line,t);
        WRITE (my_line,d_space);
        WRITELINE(outf,my_line);
END IF;
Tr:=Lr/rr;
a:=(lm*lm-lr*ls);
vdsr1 <= vdsr;

vqsr1 <= vqsr;
ids1:=(rs*lr*idss-wrr*lm*lm*iqss-rr*lm*idrr-wrr*lr*lm*iqrr-
r*vdsr1)/a;
iqs1:=(wrr*lm*lm*idss+rs*lr*iqss+wrr*lr*lm*idrr-rr*lm*iqrr-
r*vqsr1)/a;
idr1:=-(rs*lm*idss-wrr*lm*ls*iqss-rr*ls*idrr-wrr*lr*ls*iqrr-
lm*vdsr1)/a;
iqr1:=-(wrr*lm*ls*idss+rs*lm*iqss+wrr*lr*ls*idrr-rr*ls*iqrr-
lm*vqsr1)/a;
idss:=idss+(ids1*dt);
iqss:=iqss+iqs1*dt;
idrr:=idrr+idr1*dt;
iqrr:=iqrr+iqr1*dt;
Te:=p*(3.0/2.0)*lm*(iqss*idrr-idss*iqrr);
wr1:=(Te-Tl)/jr;
wrr:=wrr+wr1*dt;
thetar:=thetar+wrr*dt;
t:=t+dt;
-- End of electrical+mechanical model
IF next_step<500 THEN
next_step<=next_step+1 AFTER deltat;
ELSE
next_step<=1 AFTER deltat;
END IF;
ids<=idss;
iqs<=iqss;
wr<=wrr;
END PROCESS;
END arch_motor;

CONFIGURATION conf_motor OF motor IS
  FOR arch_motor
  END FOR;
```

Figure 5.5 shows a flow diagram for the section of a computer program, which simulates the electromechanical behaviour of the induction motor.

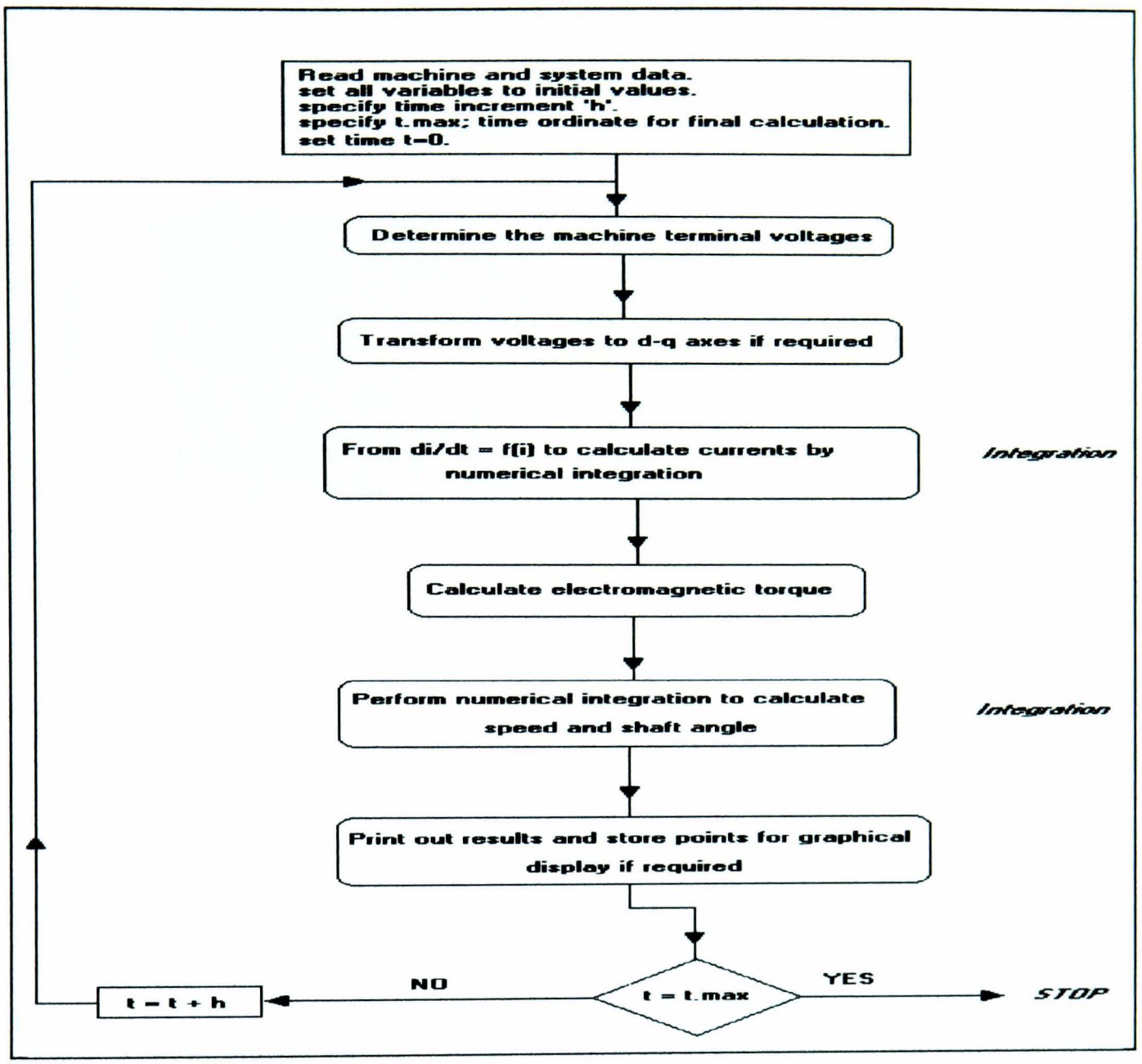


Figure 5.5 - *General flow diagram for digital-computer numerical solution of electromechanical performance*

To test the model, a simulation using VHDL is carried out, based on the algorithm shown on Figure 5.5. The chosen parameters for modelling are [4]:

$R_s = 5.9\Omega$.	Stator resistance.
$R_r = 4.62\Omega$.	Rotor resistance.
$L_s = 0.831H$	Stator inductance.
$L_m = 0.809H$	Magnetizing inductance.
$L_r = 0.833H$	Rotor inductance.
P $= 2$	Number of poles.
$J_r = 0.001$ Kg.m^2	Moment of inertia.

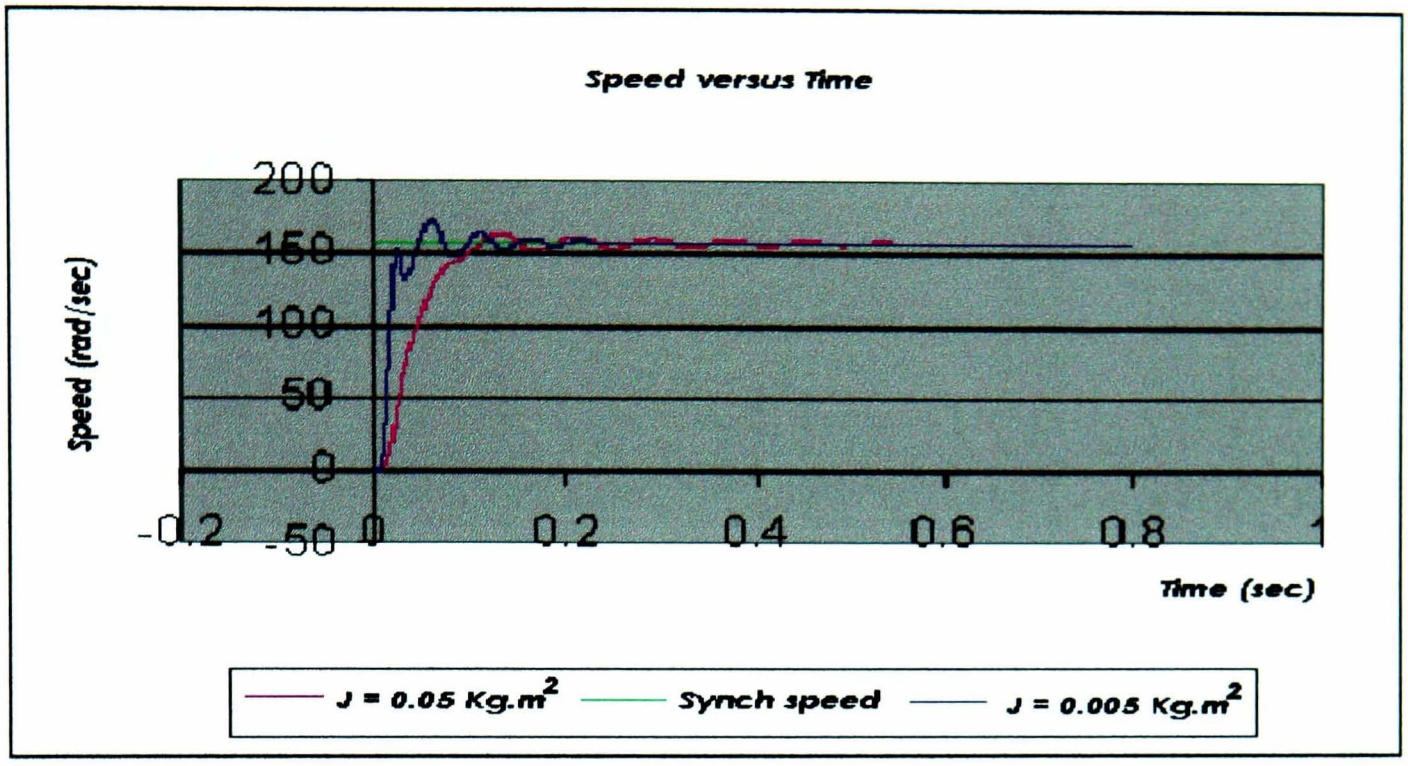


Figure 5.6 - *Speed plotted versus time*

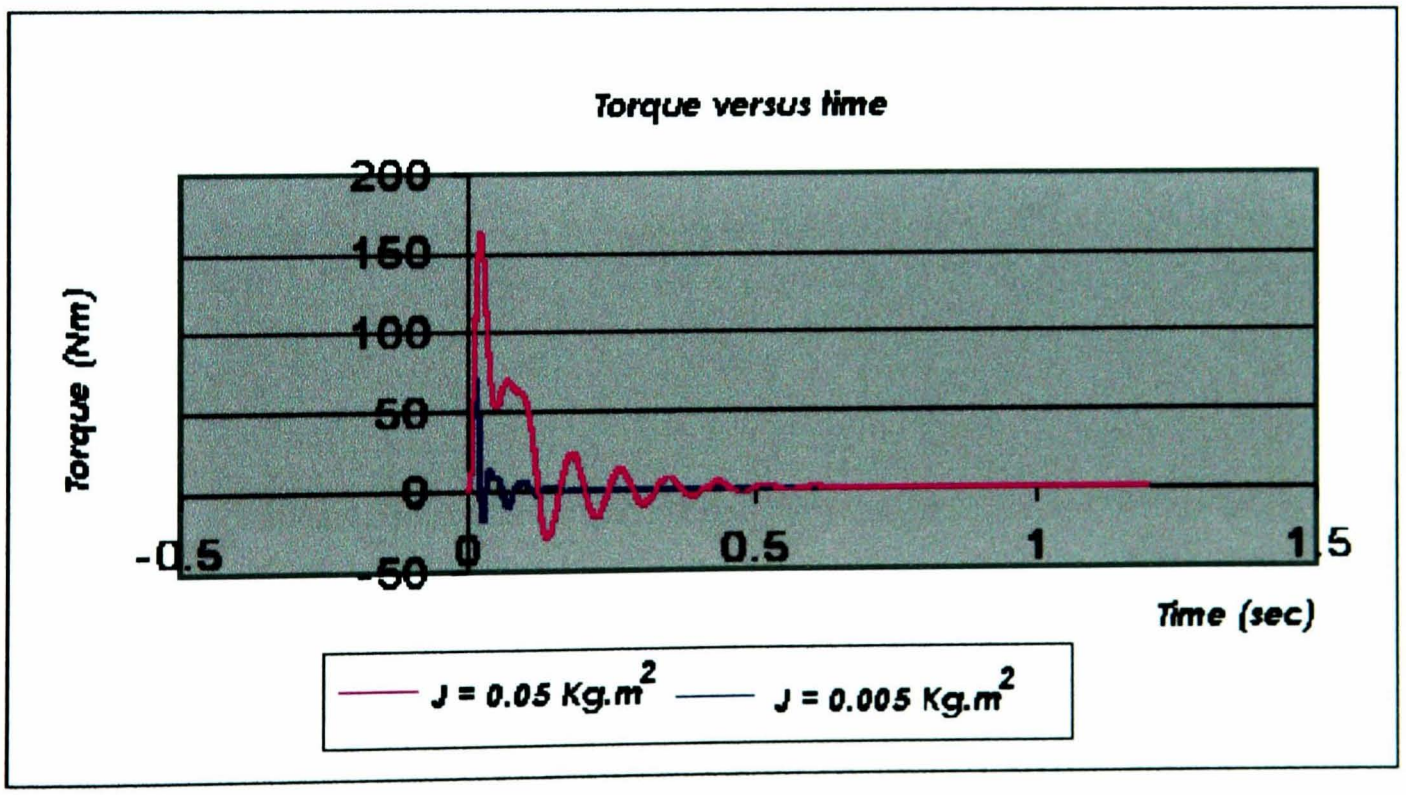


Figure 5.7 - *Torque plotted versus time at start up*

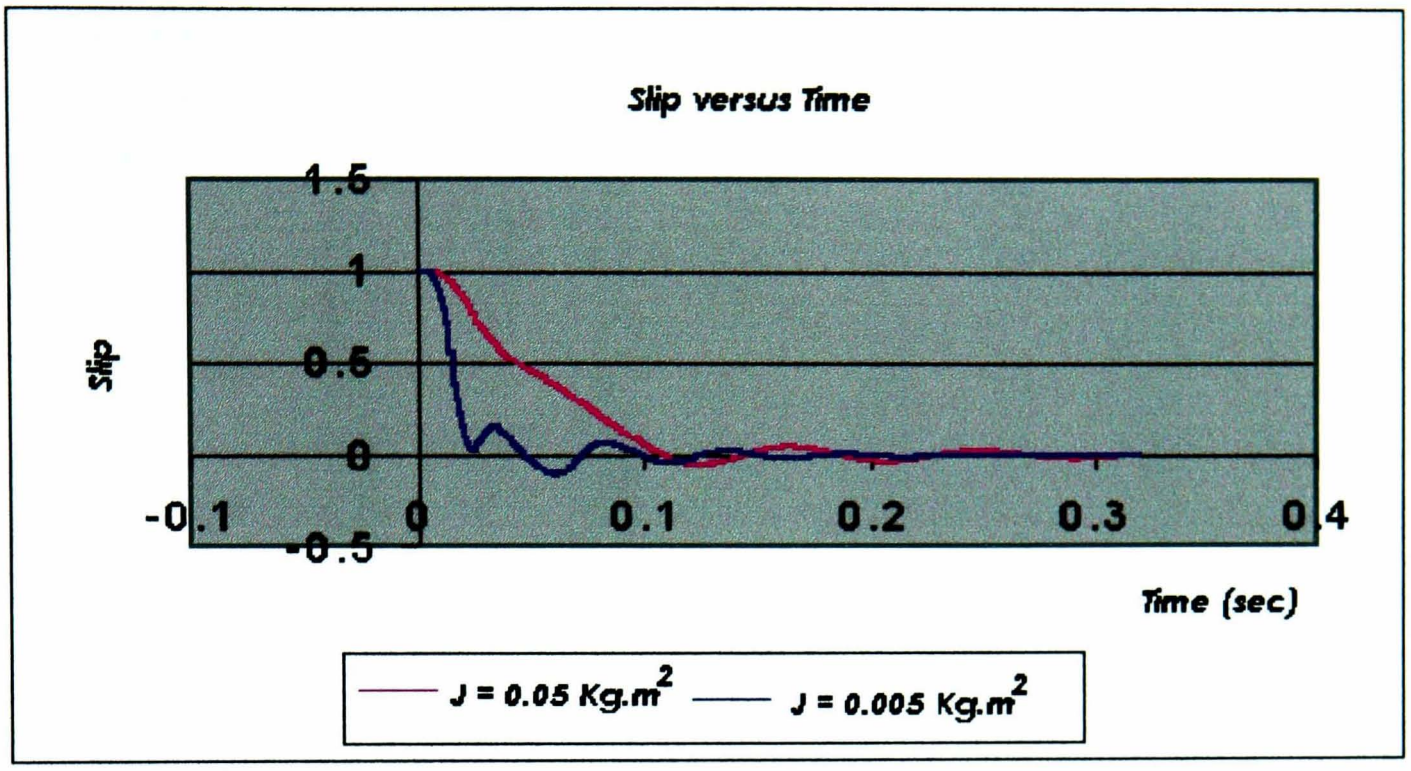


Figure 5.8 - *Slip plotted versus time*

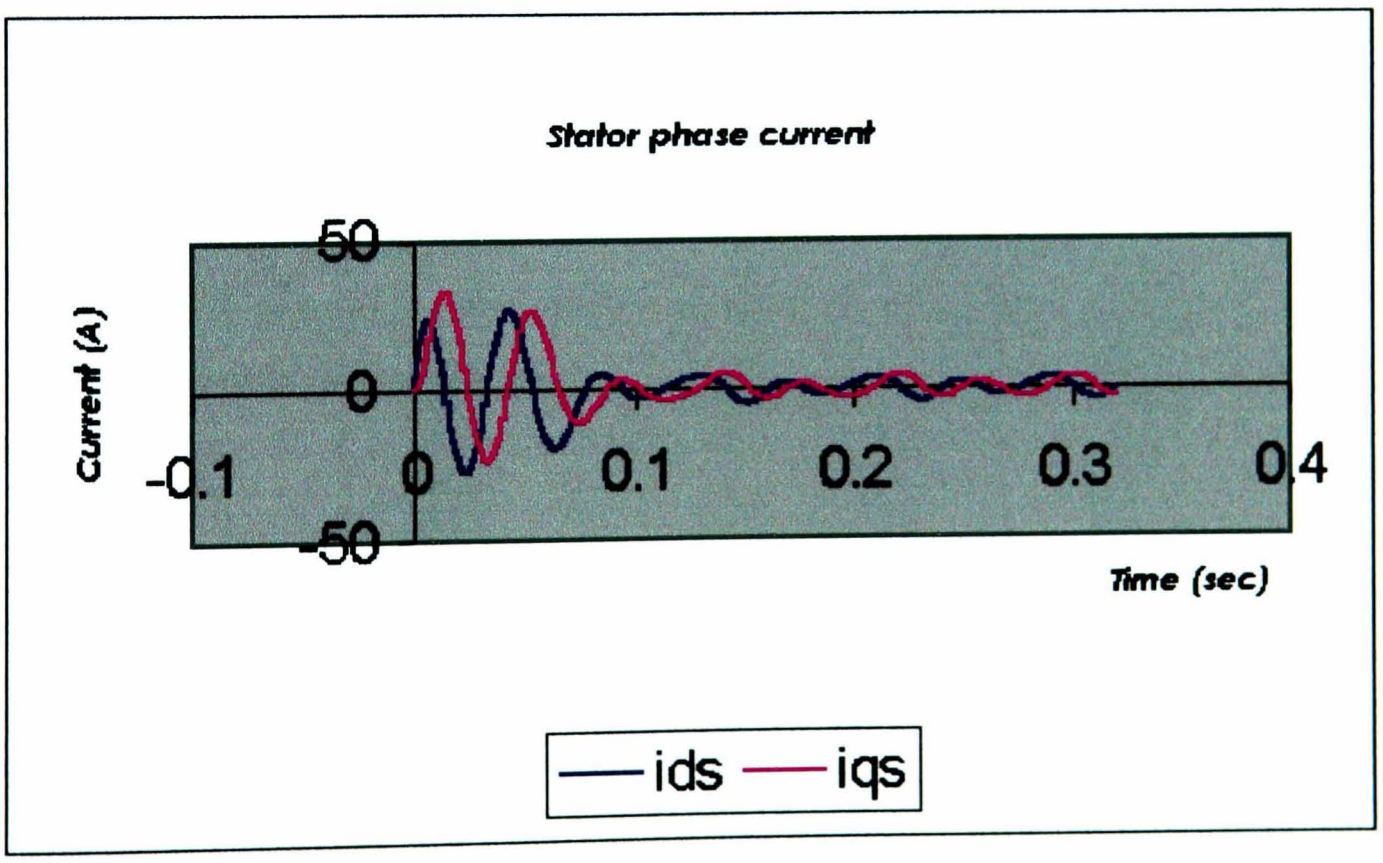


Figure 5.9 - *Stator phase current*

Simulation was carried out for two different values of inertia, as it can be seen from (Figure 5.6 to Figure 5.10) with an integration step size of (0.0000001) using the Euler method [61] to solve the equations.

The induction motor is, of course, an electromechanical device, so the model requires expressions for the electromagnetic torque and speed of the machine.

In Figure 5.6, where speed plotted against time the oscillatory transient behaviour of the machine can be clearly seen. Comparing the two speed-time plots corresponding to the two different values of inertia shows that the motor model behaves as expected. When $J_r = 0.05$ Kg.m^2, the higher inertia causes the motor to reach the oscillation period after a delayed time with respect to the speed graph for J_r=0.005 Kg.m^2. When J_r=0.005 Kg.m^2, the synchronous speed is $\omega_s = \frac{2\pi f}{p} = \frac{2\pi * 50}{2} = 157.08$ *rad/s*, the simulated steady state rotor speed is $\omega_r = 157.0407$ *rad/s* and slip is $s = \frac{\omega_s - \omega_r}{\omega_s} = 0.00025$.

The plot of slip against time is shown in Figure 5.8. It can be seen that at starting the value for slip is s=1, which corresponds to $\omega_r = 0$ *rad/s* and then the rotor goes through a transient interval before settling at the steady state slip of 0.00025. It can be concluded from Figure 5.8 that if the rotor of a motor is locked so that it can not move, then the rotor induced current will have the same frequency as the stator and the slip s=1. On the other hand, if the rotor turns at synchronous speed, the frequency on the rotor will be zero and the slip s=0.

One advantage of using models is the ability to look at the range of variables and their influence on performance. Figure 5.9 shows the stator current in the *dq-axis* as a function of time during acceleration. It can be seen from the graph that the amplitude is initially high (id_s=20A, iq_s=30A). However, within (0.008 *sec)* the rotor speed reaches nearly the synchronous speed and the motor starts to draw only a small amount of current.

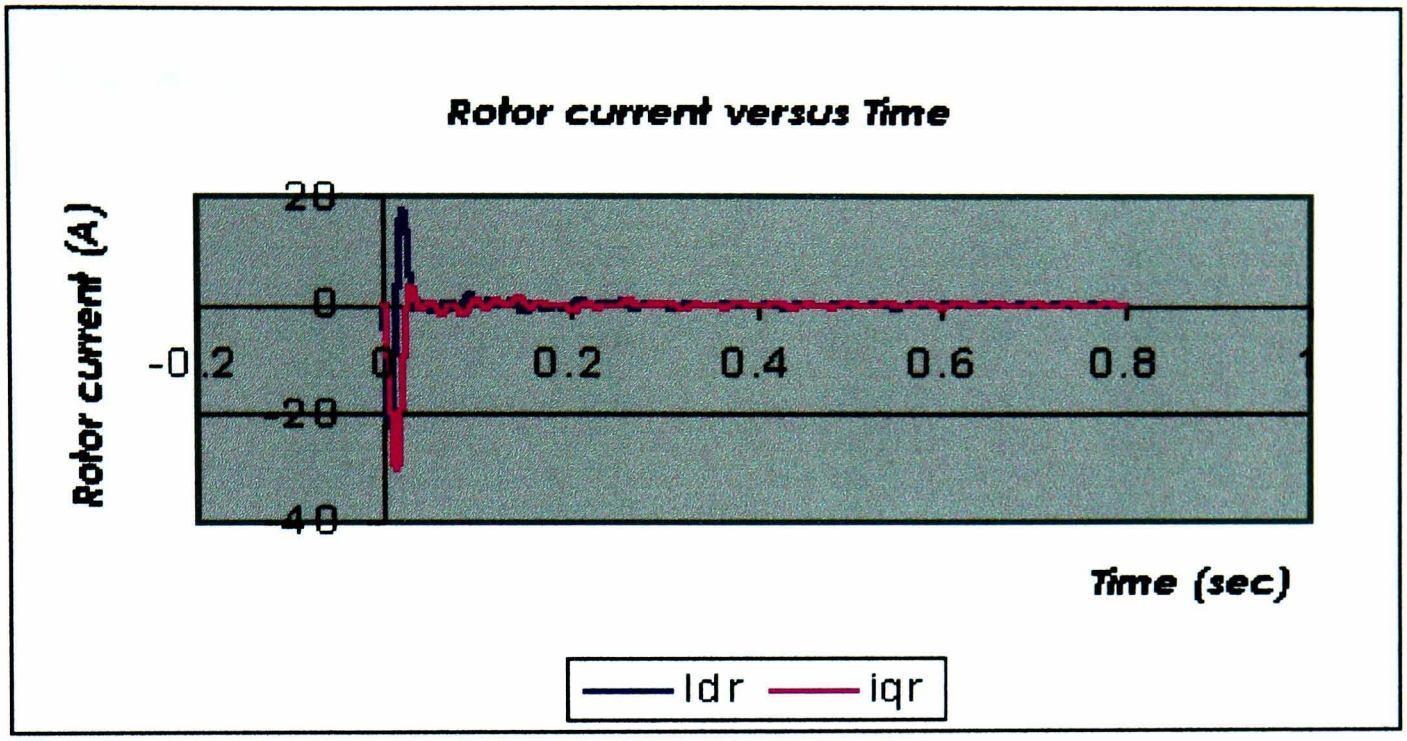


Figure 5.10 - *Rotor current plotted against time*

It would be difficult, if not impossible, to measure the rotor currents in an induction motor, but their values can be obtained through calculation or simulation. Figure 5.10 shows the rotor current in the *dq-axis*. It is clearly shown that at start the rotor currents are high and are equal to (id_r=20A, iq_r=-30A) and, as the motor accelerates, the currents in the rotor bars drop approximately to zero at (0.02 *sec*).

5.4 The Rotor Flux Oriented Vector Control.

A vector control algorithm normally determines the asynchronous motor to have characteristics and behaviour similar with those to a separately excited d.c. motor, where the torque is controlled by the armature current and the flux by an excitation current [62]. Therefore in vector control induction motor the control algorithm is used independently to control the two components of the currents, i.e. the component producing torque and flux. This is achieved by referencing the voltages and currents in a special rotating reference frame which can be fixed either to the rotor flux space phasor, the stator flux space phasor or the magnetizing flux space phasor.

The xy-axes reference frame shown in Figure 5.11 rotates at the speed of the rotor flux-linkage space phasor ω_{mr}. where $\omega_{mr} = \dfrac{d\rho_r}{dt}$ and ρ_r = phase angle of the rotor flux linkage space phasor with respect to the direct axis of the stator reference frame.

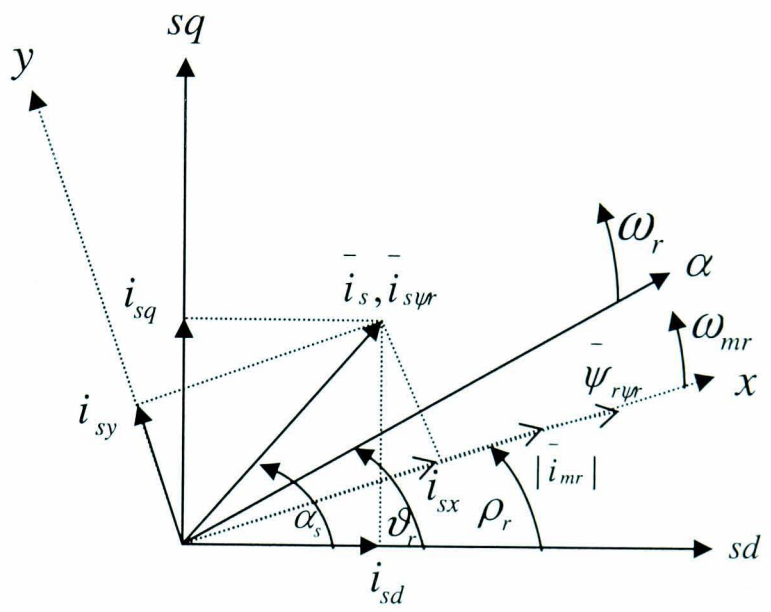


Figure 5.11 - *Stator current and rotor flux linkage space phasors in the stationary reference frame and in the special reference frame fixed to the rotor flux linkage space phasor*

The electromagnetic torque is defined as :

$$T_e = \frac{3}{2} p \frac{L_m}{L_r} \psi_{rx} i_{sy} \qquad \textbf{5-31}$$

Since $\psi_{ry} = 0$, meaning that the rotor linkage flux space phasor has only a x-axis component, the rotor linkage flux space phasor is given as:

$$|\bar{\psi}_{r\psi r}| = \psi_{rx} = L_m |\bar{i}_{mr}| \qquad \textbf{5-32}$$

Therefor substituting equation **5-32** into equation **5-31** yields:

$$T_e = \frac{3}{2} p \frac{L_m^2}{L_r} |\bar{i}_{mr}| i_{sy} \qquad \textbf{5-33}$$

It can be shown from equation **5-33** that the torque can be controlled by independently controlling $|\bar{i}_{mr}|$ and i_{sy}.

In the special rotor-flux oriented (xy) reference frame, the space phasor of the stator currents can be expressed in terms of the space phasor of the stator currents established in the stationary reference frame as:

$$i_{sx} = i_{sD}\cos\rho_r + i_{sQ}\sin\rho_r$$
$$i_{sy} = -i_{sD}\sin\rho_r + i_{sQ}\cos\rho_r \qquad \textbf{5-34}$$

i_{sx} is the flux producing component and i_{sy} is the torque producing component. These two components must be independently controlled to achieve a rotor flux oriented control.

5.4.1 The Flux Model.

In the flux model modulus and phase angle of the rotor flux phasor must be calculated to obtain ω_{mr} and $|\bar{i}_{mr}|$. The space phasor of the rotor magnetizing currents expressed in the magnetizing flux oriented reference frame can be found from the following equation [58]:

$$T_r\frac{d|\bar{i}_{mr}|}{dt} + |\bar{i}_{mr}| = i_{sx} \qquad \textbf{5-35}$$

$$\omega_{mr} = \omega_r + \frac{i_{sy}}{T_r|\bar{i}_{mr}|} \qquad \textbf{5-36}$$

Where:

ω_{mr} is the angular speed of the rotor flux oriented reference frame.

i_{sx}, i_{sy} are the instantaneous values of the direct-and quadrature axis stator current components respectively, and expressed in the general reference frame (flux and torque producing stator current components respectively).

$|\bar{i}_{mr}|$ modulus space phasor of the rotor magnetizing currents expressed in the magnetizing flux oriented reference frame.

ω_r angular rotor speed.

ω_{sl} angular slip frequency.

Based on equation **5-35** and **5-36**, a rotor flux can be determined, as shown by the block diagram Figure 5.12. The angular slip frequency of the rotor flux is $\omega_{sl} = \dfrac{i_{sy}}{T_r \mid i_{mr} \mid}$.

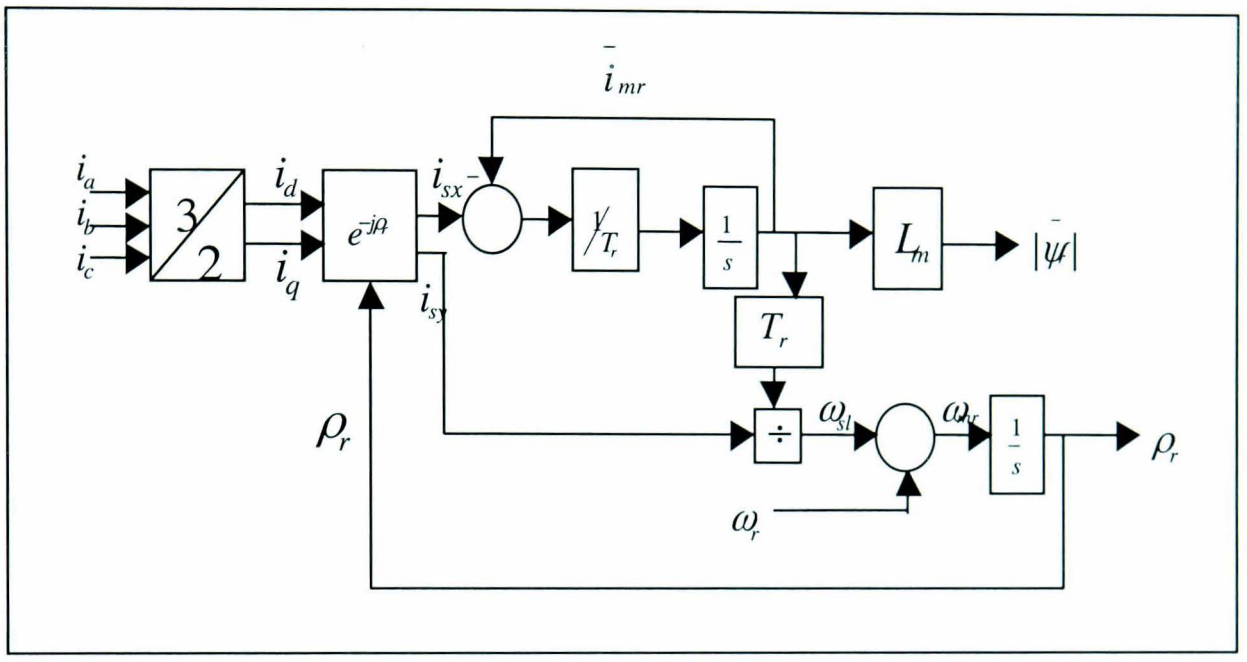


Figure 5.12 - *Flux models in the rotor flux oriented reference frame*

The simulation of the rotor flux oriented vector control is based on the following differential equations [58]:

$$T_s' \frac{di_{sx}}{dt} + i_{sx} = \frac{v_{sx}}{R_s} + \omega_{mr} T_s' i_{sy} - (T_s - T_s') \frac{d\,\bar{i}_{mr}}{dt}$$

$$T_s' \frac{di_{sy}}{dt} + i_{sy} = \frac{v_{sy}}{R_s} - \omega_{mr} T_s' i_{sx} - (T_s - T_s') \omega_{mr} \mid \bar{i}_{mr} \mid \qquad \textbf{5-37}$$

where :

$T_s = \dfrac{L_s}{R_s}$ is the stator time constant

$T_s' = \dfrac{L_s'}{R_s} = \dfrac{(L_s - L_m^2 / L_r)}{R_s}$ is the stator transient time constant of the machine .

L_s' is the stator transient inductance.

It can be seen from equation **5-37** that there is unwanted coupling between the stator circuits on the two axes. For the purposes of rotor flux oriented control, it is important that the direct axis stator current i_{sx} (rotor flux producing component) and the quadrature axis stator current i_{sy} (torque producing component) are independently controlled.

If $|\bar{i}_{mr}|$ is constant then $\frac{d}{dt}|\bar{i}_{mr}|=0$ and equations **5-37** become:

$$L_s^{'}\frac{di_{sx}}{dt}+R_s i_{sx}=v_{sx}+\omega_{mr}L_s^{'}i_{sy}$$

$$L_s^{'}\frac{di_{sy}}{dt}+R_s i_{sy}=v_{sy}-\omega_{mr}L_s^{'}i_{sx}-(L_s-L_s^{'})\omega_{mr}|\bar{i}_{mr}| \qquad \textbf{5-38}$$

It follows from equation **5-38** that the stator current components can be independently controlled if the decoupling rotational voltage components are added to the output of the current controllers which control i_{sx} and i_{sy} respectively.

The rotational voltage components are:

$$v_{dx}=-\omega_{mr}L_s^{'}i_{sy} \qquad \textbf{5-39}$$

$$v_{dy}=\omega_{mr}L_s^{'}i_{sx}+(L_s-L_s^{'})\omega_{mr}i_{mr} \qquad \textbf{5-40}$$

The voltages of the output controllers are:

$$\hat{v}_{sx}=R_s i_{sx}+L_s^{'}\frac{di_{sx}}{dt} \qquad \textbf{5-41}$$

$$\hat{v}_{sy}=R_s i_{sy}+L_s^{'}\frac{di_{sy}}{dt} \qquad \textbf{5-42}$$

The required decoupling circuit is shown in Figure 5.13

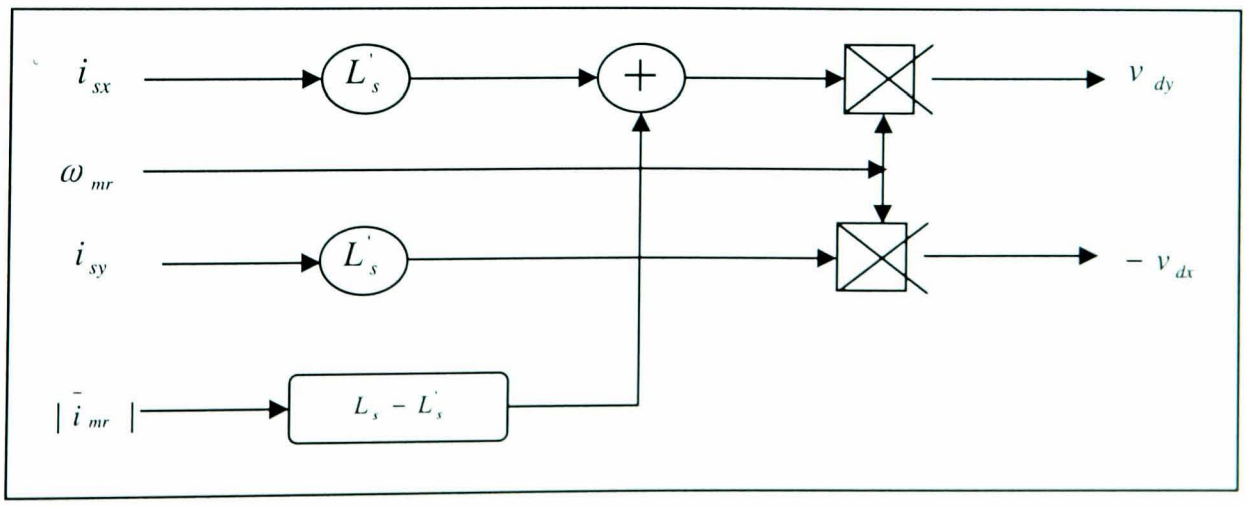


Figure 5.13 - *Decoupling circuit*

The currents on the x and y axis can be obtained from the following matrix:

$$\begin{bmatrix} i_{sx} \\ i_{sy} \end{bmatrix} = \begin{bmatrix} \cos\rho_r & \sin\rho_r \\ -\sin\rho_r & \cos\rho_r \end{bmatrix} \begin{bmatrix} i_{ds} \\ i_{qs} \end{bmatrix} \qquad \textbf{5-43}$$

5.4.2 Control Strategy

Control of induction machines is associated with complexity and technological challenge. Simple controllers exhibit non linear characteristics and, in general, operate satisfactory only over a limited speed range. Variable frequency control involves the use of an inverter connected to the machine and, until recently, the design of the inverter and machine system were separate activities. However, recently, the trend has been towards the integration of the design process, thereby facilitating the study of the effects of the inverter output waveform on the dynamic load and vice versa. To do this effectively, appropriate models of both systems are pre-requisite. Dynamical systems require to be studied on a transient basis and consequently modelling of the induction machine using generalised machine theory is employed.

Advanced control techniques, including vector control, have been incorporated into induction machine drive systems. The objective is to develop fast acting controllers, which react to load disturbances and input reference variations as fast as d.c. machine systems. Several important elements need to be constructed, an important one being a PWM controller.

5.4.2.1 Pulse Width Modulation

The Pulse Width Modulation technique is used to generate the required voltage or current to feed the motor. This method is increasingly used for a.c. drives with the condition that the output harmonic content is as small as possible [63]. PWM is currently the most widely used technique of inverter control and has received considerable attention in the last two decades. The PWM switching scheme essentially involves the strategic variation of the ON and OFF timing periods of each pair of switches in the inverter.

This produces a PWM waveform that contains a series of pulses which have the same voltage level but different widths, as shown by the waveform in Figure 5.15.

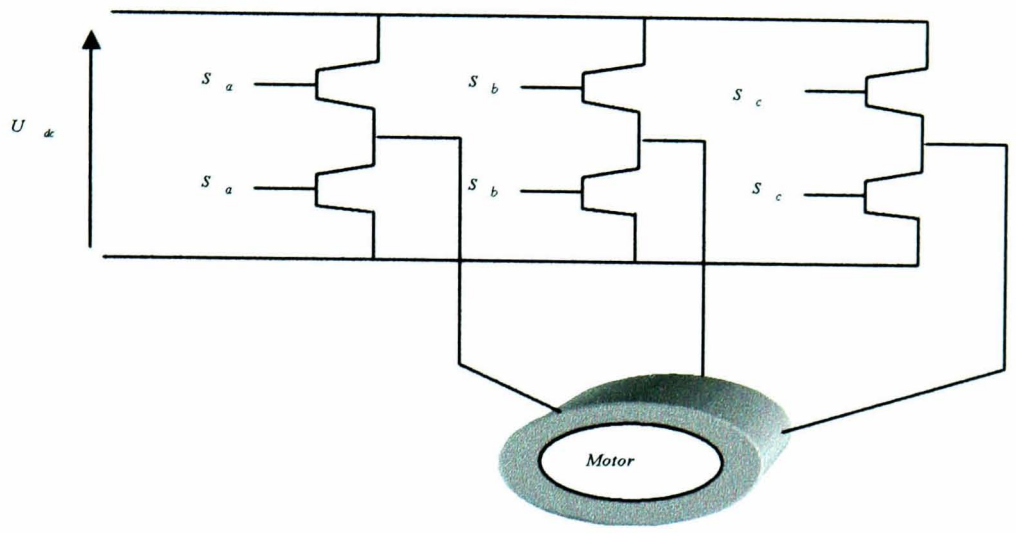


Figure 5.14 - *A three phase inverter*

The fundamental component of the PWM switching pattern V_{PWM} in Figure 5.15 is a sinewave, which is the required output voltage. To obtain the switching pattern, a sinusoidal signal $V_{Control}$ is compared with high frequency triangular carrier wave V_{tri}. This form of PWM control is sometimes called sinusoidal-PWM in order to explicitly differentiate it from other forms of PWM control schemes.

In sinusoidal-PWM control schemes, there are two characteristic ratios which are important factors in the design of the controllers.

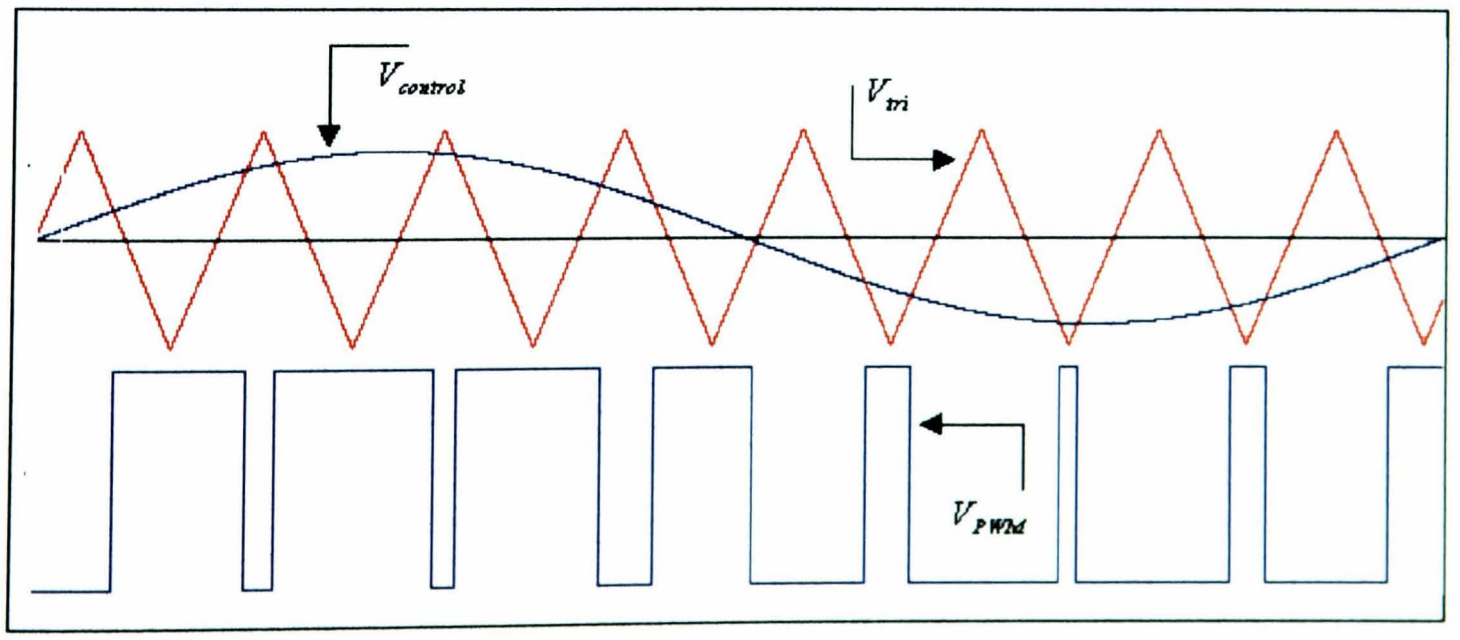


Figure 5.15 - *Pulse Width Modulation*

The amplitude modulation ratio m_a is defined as the ratio of the peak amplitude of the control $V_{control}$ to the peak amplitude of the triangular carrier signal $V_{tri.}$

$$m_a = \frac{V_{control}}{V_{TRI}} \qquad \textbf{5-44}$$

The frequency modulation ratio m_f is defined as the ratio of the triangular carrier frequency f_{tri} to the ratio of the control signal frequency $f_{control}$.

$$m_f = \frac{f_{tri}}{f_{control}} \qquad \textbf{5-45}$$

5.4.2.2 Space Vector Modulation .

The Space Vector Pulse Width Modulation (SVPWM), beside being a technique which is perfect for digital implementation, offers a better utilisation of the d.c. link voltage and a lower harmonic content, particularly at high modulation indices.

The Space Vector PWM technique is defined by the approximation of V_{ref}, the reference voltage, which can be defined as:

$$V_{ref} = v_{dref} + jv_{qref} \qquad \textbf{5-46}$$

For the vector control scheme, V_{ref} is given by v_{dref} and v_{qref}. As the desired output is a set of three phase sinusoidal voltages, v_{ref} is a vector rotating around the origin of the *dq-axis* plane. Its frequency is given by that of the desired three phase voltages.

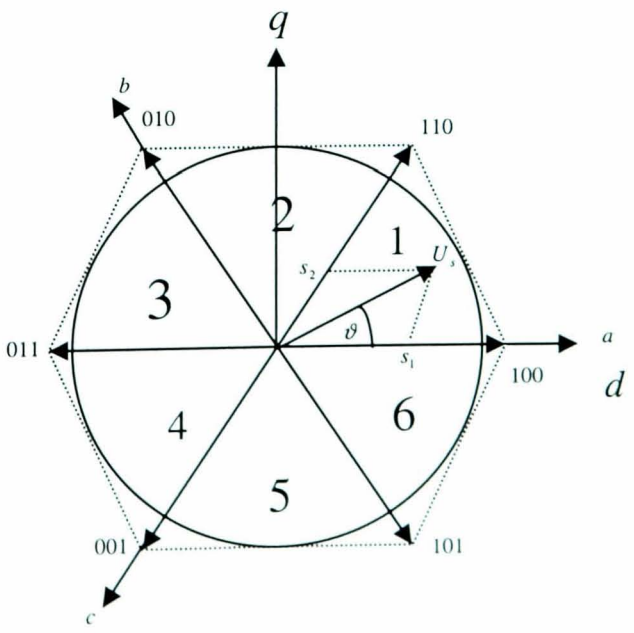


Figure 5.16 - *Space vector combination*

For a better understanding of the space vector process and in order to represent the switching state of the inverter in Figure 5.14, the switching function of each phase is defined as '1' when the upper transistor of that phase is ON and '0' when the upper transistor is OFF. Each set of three bits in the corners of the hexagon in Figure 5.16 represents the state of the switches on phase *a, b* and *c* respectively. In the space vector theory, the motor voltage vector is approximated by a combination of 8 switching patterns of the 6 power transistors represented in Figure 5.16. A third PWM generation method uses pre-calculated switching times so that specific harmonics are eliminated from the signal in order to improve system operation.

5.4.3.2 The FOC Algorithm Structure

Many different vector control structures are possible for a.c. induction motor depending on the desired performance level and the acceptable implementation. Both direct and indirect vector controller structures are possible, depending on whether or not there is a direct measurement or estimation of the flux quantity, to which the current must be oriented.

Most often there is no direct measurement of either the produced torque or flux so that the control is implemented by a closed loop current regulation whose references are derived from a feed forward control structure for the induction motor [64], [65], known as the Indirect Rotor Field Oriented Controller. Such a system is illustrated in Figure 5.17.

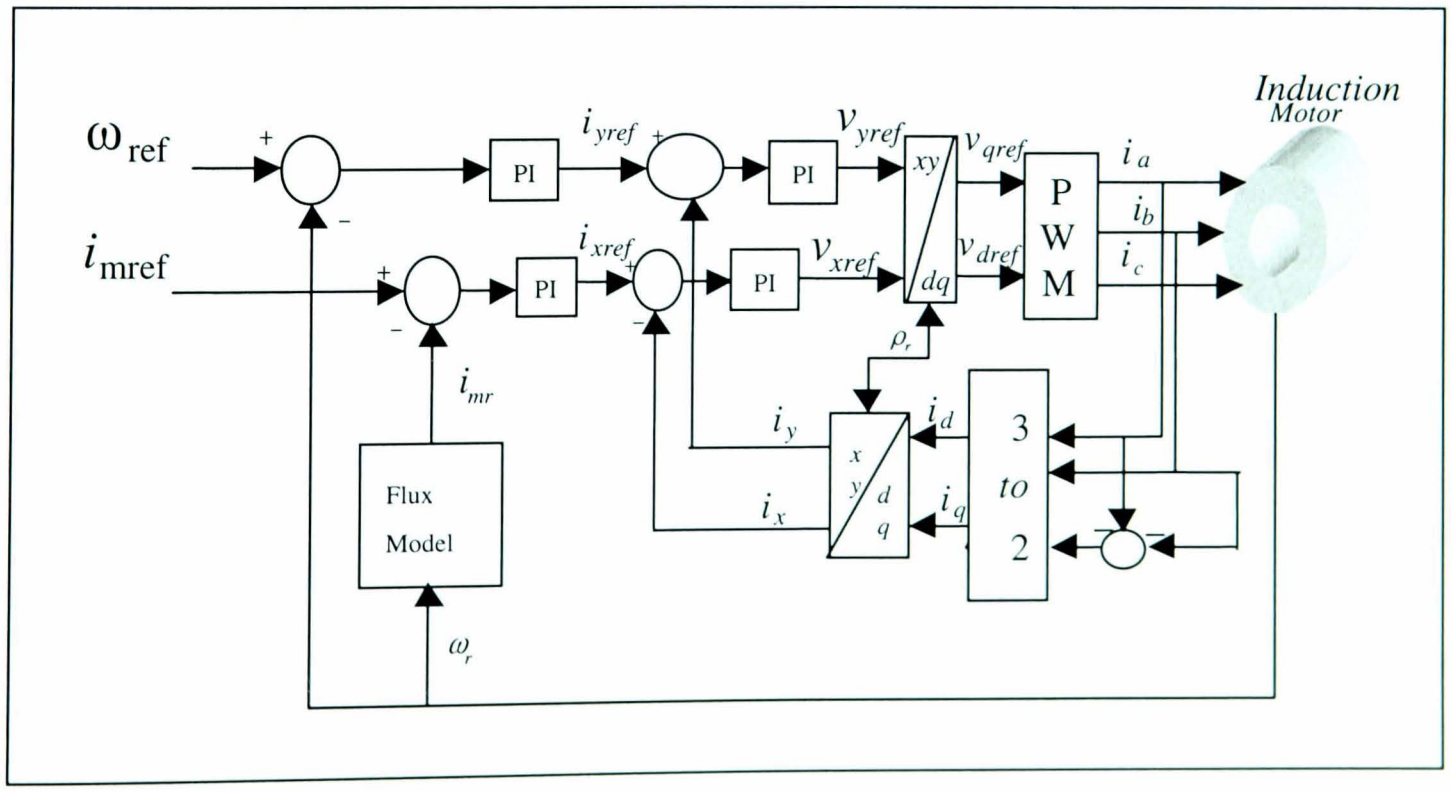


Figure 5.17 - *Control principle scheme of a.c. machine*

The Induction Machine is supplied by a voltage-source PWM inverter. The output voltages of the inverter are controlled by a pulse width modulation technique.

The flux model shown in Figure 5.12 generates the angle ρ_r, which is used in the transformation blocks. Furthermore, the flux model is used to obtain the angular speed of the rotor flux ω_{mr} and the modulus of the magnetizing current $|\bar{i}_{mr}|$, since these are also used in the decoupling circuit illustrated in Figure 5.13. The modulus of the rotor magnetizing current is also used to obtain the electromagnetic torque in accordance with equation **5-33**.

Vector control works on the principle of measuring two phase currents i_a, i_b and then the third one i_c is calculated from $i_c = -(i_a + i_b)$ and then the three components are transformed into two current components in the (d,q) rotating frame.

The speed controller, of type PI, provides the reference torque (t_{eref}). The torque controller, again of PI type, gives the reference value of the quadrature axis stator current in the rotor flux oriented reference frame (i_{syref}). The reference signal $|i_{mref}|$ is compared with the actual value of the rotor magnetizing current and the error serves as input to the flux controller which, is a PI controller. Its output is the direct axis stator current reference (i_{sxref}). The error signals ($i_{sxref} - i_{sx}$) and ($i_{syref} - i_{sy}$) are the inputs to the respective current controllers. The outputs of these controllers are added to the corresponding outputs of the decoupling circuits. Thus, the direct and the quadrature axis reference stator voltages v_{xref} and v_{yref} are obtained. They have then to be transformed by $e^{j\rho_r}$ to obtain the two axis reference stator voltages in the stationary reference frame (v_{qref}, v_{dref}) and are subsequently subject to 2-phase to 3-phase transformation.

5.4.3 Simulation Results.

A series of computer simulation are conducted on the controller to analyse its performance before proceeding to hardware implementation. The VHDL model for the complete vector controlled induction motor is configured by the entity ***Motor***. The input signals of the model are current in the ***d-q*** axis, the load torque $\mathbf{T_L}$ and the speed command ω_c.

Six output signals provide simulated information of the transistor switching state and the actual rotor speed. All data regarding the motor operation during simulation is stored in an output ASCII file. The file contains numerical data matrix format. Each line in the matrix contains the set of quantities that characterise the motor operation at certain moment in time: currents, voltages, speed and torque. Part of the VHDL code is shown below:

The VHDL code starts with the declaration of constant and signals which represent:

- Motor parameters.
- Speed controller.
- Current controller.
- Flux controller.

The structure of the code algorithm used to develop the VHDL model of the vector controlled induction motor is described below:

```
LIBRARY math;
USE math.mathtyx.all;
USE std.textio.all;
*** Electrical + mechanical model ***

ENTITY motor IS  PORT (vds,vqs,Tl: IN Real;
     ids,iqs,wr: OUT REAL);
END motor;
ARCHITECTURE arch_motor OF motor IS
{ Constant and signal declaration  };
{  Define speed controller gains   };
{  Define flux controller gains    };
{  Define currents controller gains };
BEGIN PROCESS (next_step)
 {VARIABLE declaration };

  BEGIN          { Main program  }
    { Solving differential equation for the electrical
    and mechanical model of the IM using Euler's methods}
    { Speed control loop};
    { Torque control loop};
    { iq current control loop};
    { id current control loop};
    { Flux control loop};
    { Inverse Park transformation};
    { PWM };
END PROCESS;
END arch_motor;
```

Part of the VHDL code defines a process which is one of the VHDL constructs for embodying algorithms. A *Process* statement begins with a sensitivity list (*next-step*). The next-step used here is to generate a signal that causes the process to execute as soon as a change in that signal occurs.

The process statement is completed with the reserved words (*End* process). The differential equations of the motor are represented within the process in the following manner.

```
a:=(lm*lm-lr*ls);
IF tt =1.0 then
  vdsr1 <= vds;
  vqsr1 <= vqs;
  tt <= 0.0;
  else
   vdsr1 <= vdsr;
   vqsr1 <= vqsr;
END IF;

ids1:=(rs*lr*idss-wrr*lm*lm*iqss-rr*lm*idrr-wrr*lr*lm*iqrr-lr*vdsr1)/a;
iqs1:=(wrr*lm*lm*idss+rs*lr*iqss+wrr*lr*lm*idrr-rr*lm*iqrr-lr*vqsr1)/a;
idr1:=-(rs*lm*idss-wrr*lm*ls*iqss-rr*ls*idrr-rr*lr*ls*iqrr-lm*vdsr1)/a;
iqr1:=-(wrr*lm*ls*idss+rs*lm*iqss+wrr*lr*ls*idrr-r*ls*iqrr-lm*vqsr1)/a;
```

In order to simulate continuous system on a digital computer, the independent variable must be discretized, so that differential equations become difference equations. As all the variables are functions of the independent variable t, they are defined only in discrete values of the independent variable. If simulation operates with constant independent variable increments ***dt*** (calculation interval), these points can be expressed as $t_i = t_o + i * dt$ where i =0,1,2,.....,i_{max}. Thus the simulation run begins at time t=t_o and terminates at time $t = t_{max} = t_o + i_{max} * dt$.

To simulate a differential equations using a digital computer, every simulation system must possess the following important feature:

- The equations must be solved with a numeric integration algorithm. Integration is at the heart of each simulation system. It can be realized in a very simple manner but it can also be very complicated and sophisticated, which is the case for modern numerically powerful simulation tools. Euler's algorithm is very suitable for elucidating the numerical integration procedure, illustrated on Figure 5.18.

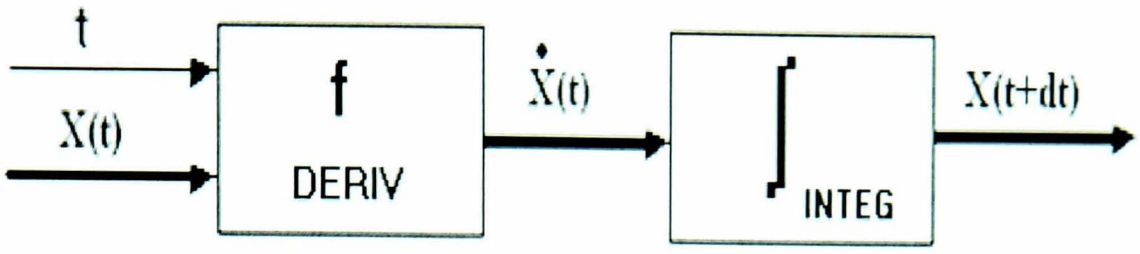


Figure 5.18 - *The basic digital simulation concept*

Once the derivative evaluation of the motor currents are achieved, the VHDL code describing the numerical integration of the currents is as follow:

*idss:=idss+ids1*dt;*

*iqss:=iqss+iqs1*dt;*

*idrr:=idrr+idr1*dt;*

*iqrr:=iqrr+iqr1*dt;*

The simulation run is executed by the integration procedure (INTEG subprogram) call after pre-simulation operations. During simulation the integration procedure requires many evaluation of state derivatives (depending on the integration algorithm). In the prescribed time instants the control is given to the OUTPUT subprogram which supplies the user with simulation results. After the simulation run some post-simulation operations are executed (e.g the processing of simulation results).

The behavioural description of the flux model using VHDL code and based on Euler numerical integration method for equations **5-35** and **5-36** is as follow:

```
imr1:=(isx-imr)/Tr;
imr:=imr+imr1*dt;
fluxr:=lm*imr;
  if imr >0.0 then
     wmr:= wrr+(isy/(Tr*imr));
  end if;
thetamr:=thetamr+wmr*dt;
```

The analysis and simulation of the control algorithm was achieved using behavioural VHDL programs. Figure 5.19 to Figure 5.22 illustrate the time response of the rotating speed, electromagnetic torque and the stator voltages and currents, when a start from zero speed is performed and at no load torque.

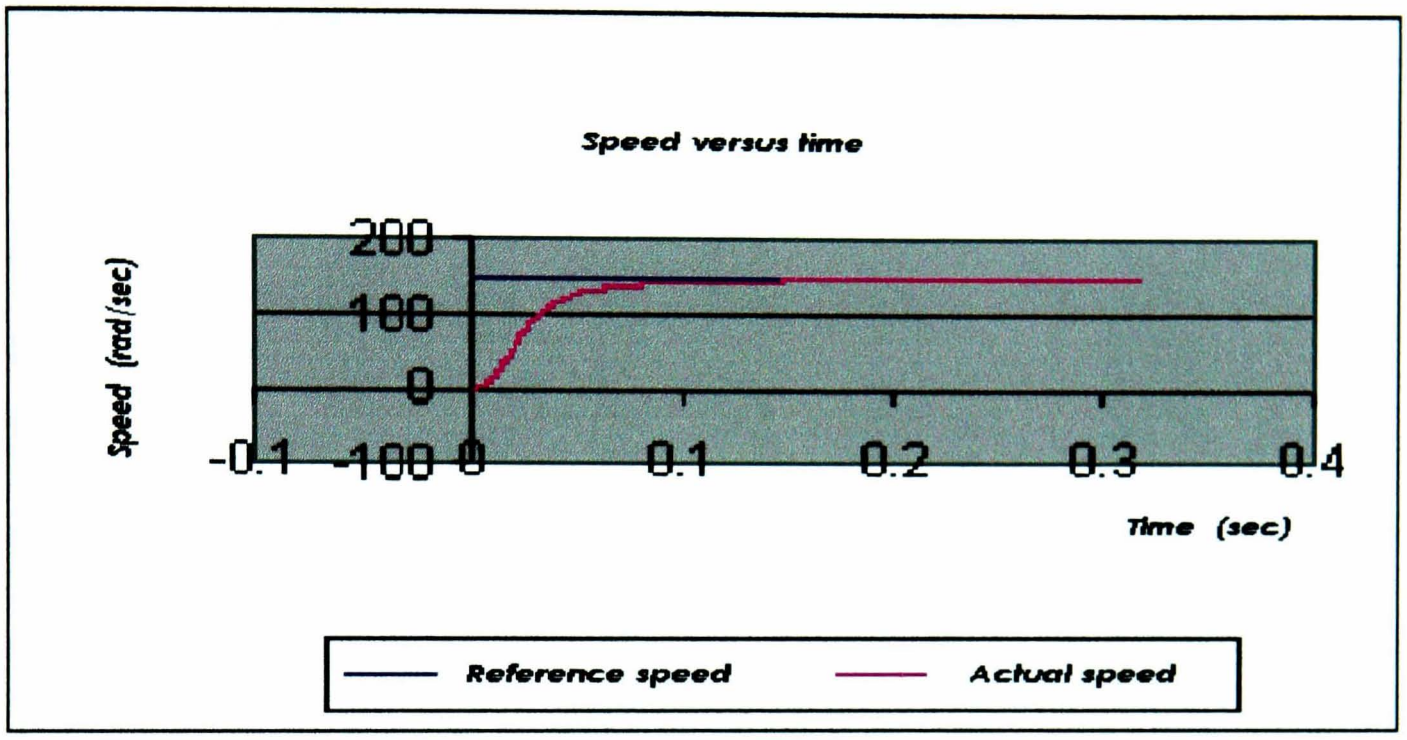


Figure 5.19 - *Speed and its reference*

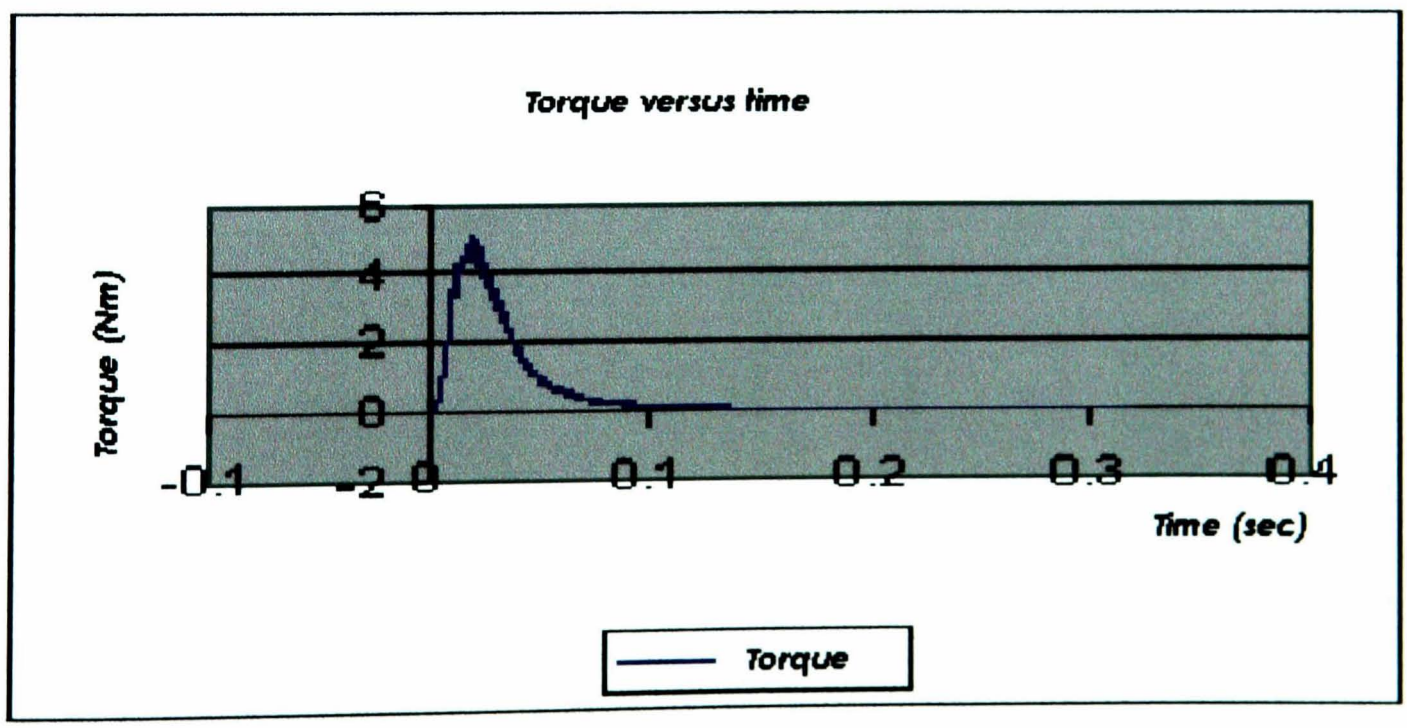


Figure 5.20 - *Electromagnetic torque versus time*

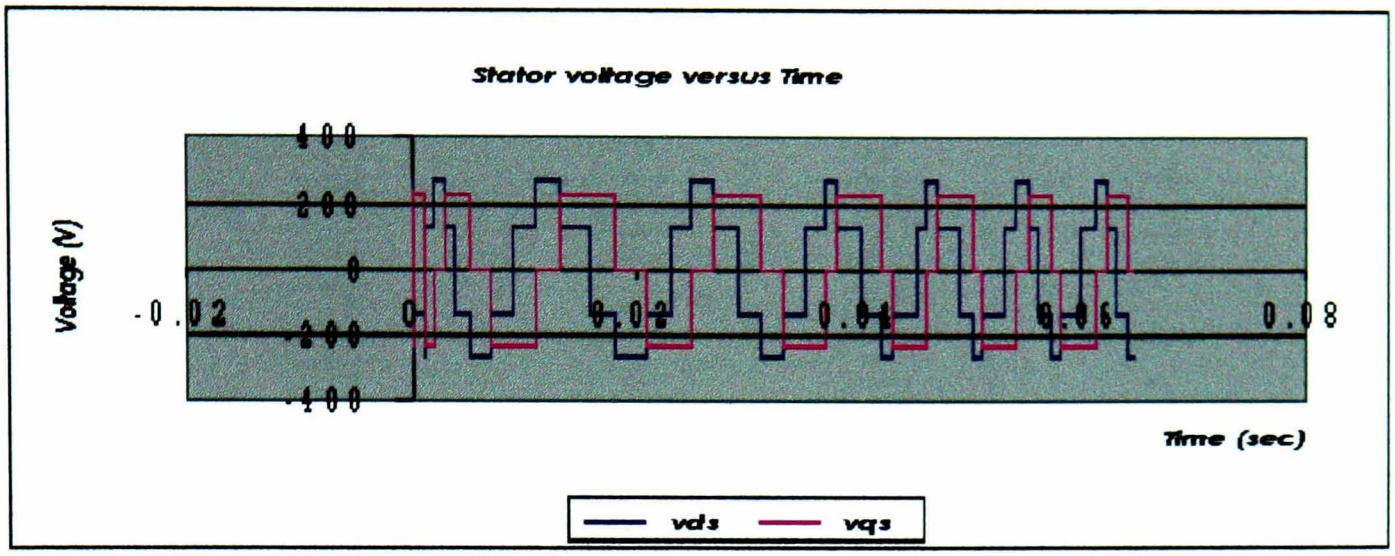


Figure 5.21 - *Current versus time*

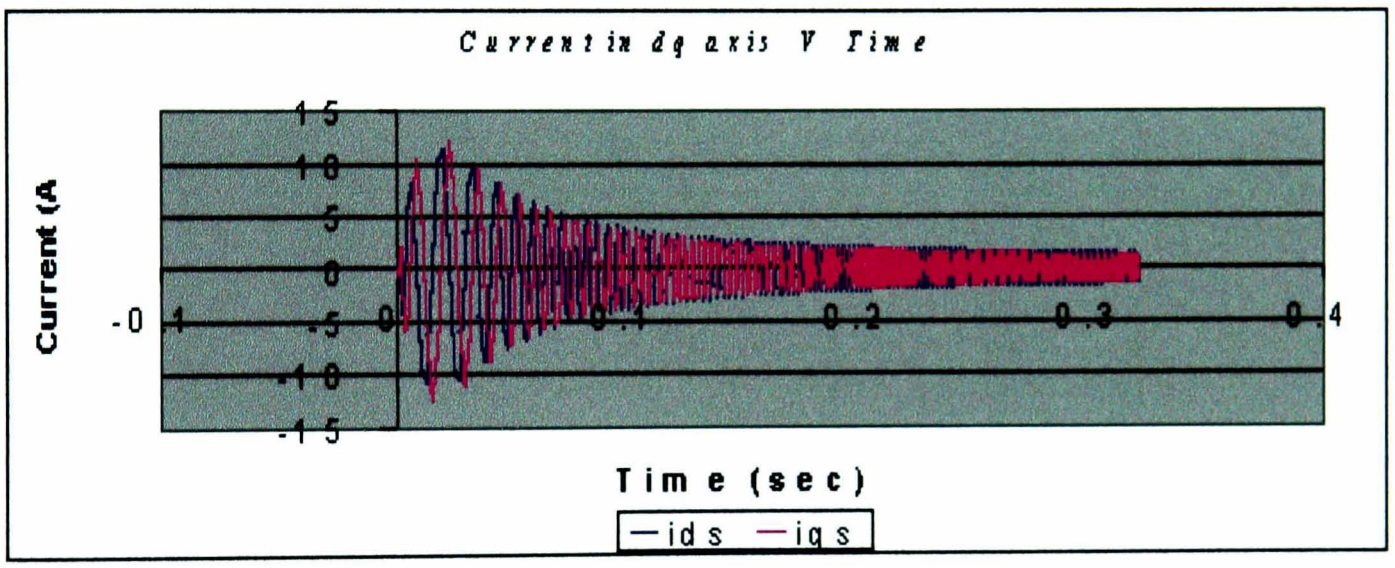


Figure 5.22 - *Stator voltages versus time*

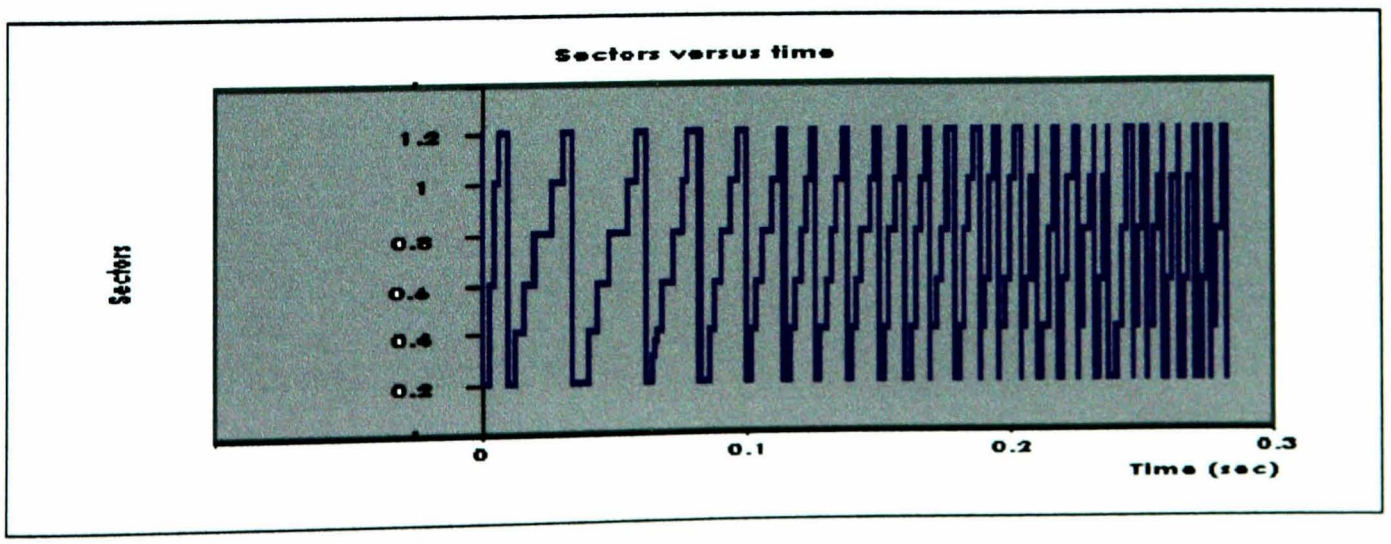


Figure 5.23 - *Sectors versus time*

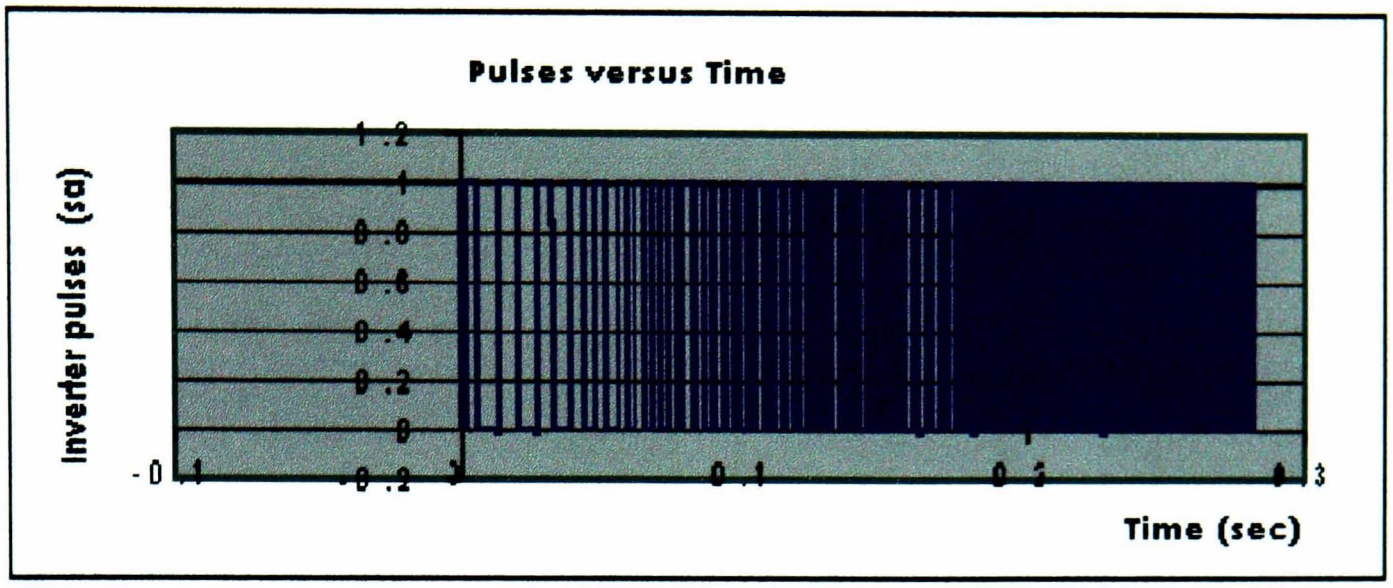


Figure 5.24 - *Pulses S_a versus time*

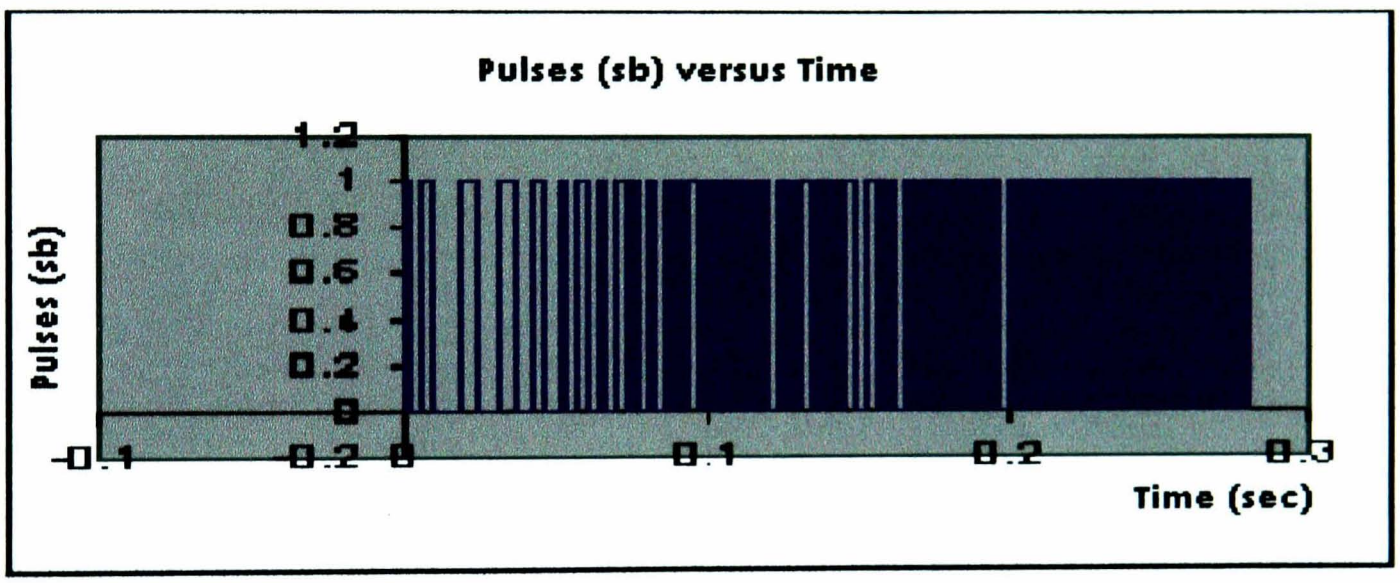


Figure 5. 25 - *Pulses S_b versus time*

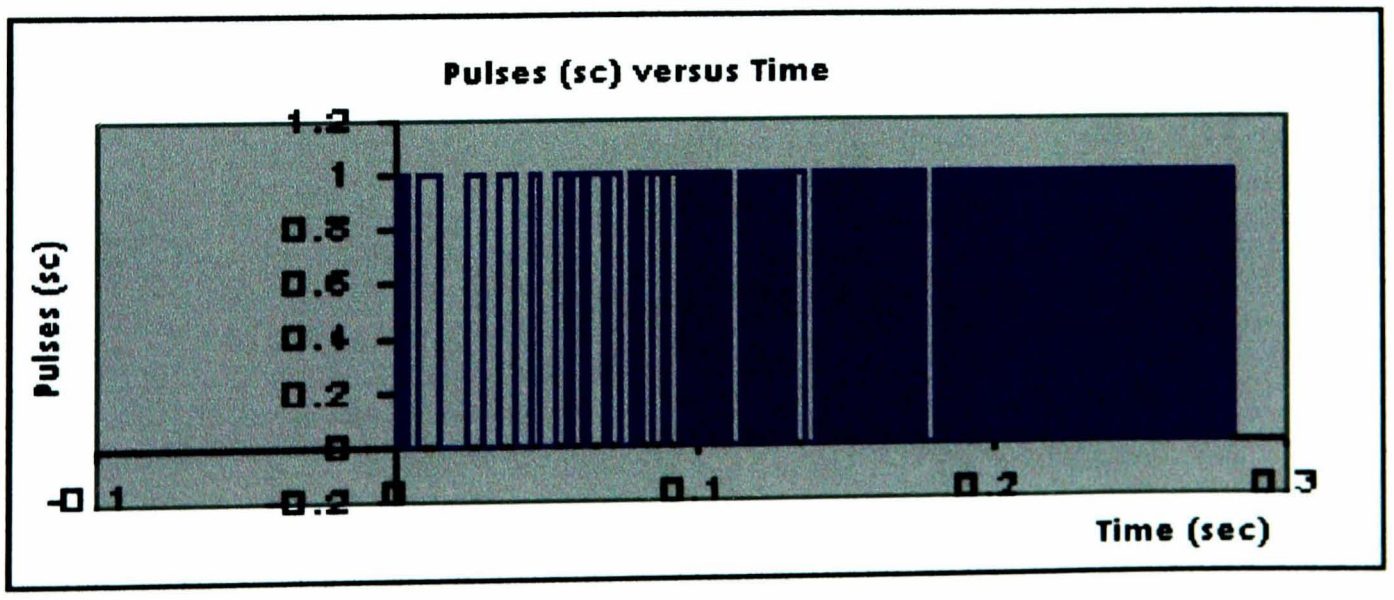


Figure 5. 26- *Pulses s_c versus time*

During the transient state, the electromagnetic torque increases to its maximum value and once the speed reaches its reference the torque drops to approximately zero. Figure 5.23 shows the sector number in accordance with Figure 5.16, plotted against time and has the sequence 1,2,3,4,5,6. The voltages v_{derf} and v_{qref} are used in the simulation to calculate the angle from which the sector is identified and therefore the correct pulses are applied as shown in Figure 5.24 to Figure 5. 26.

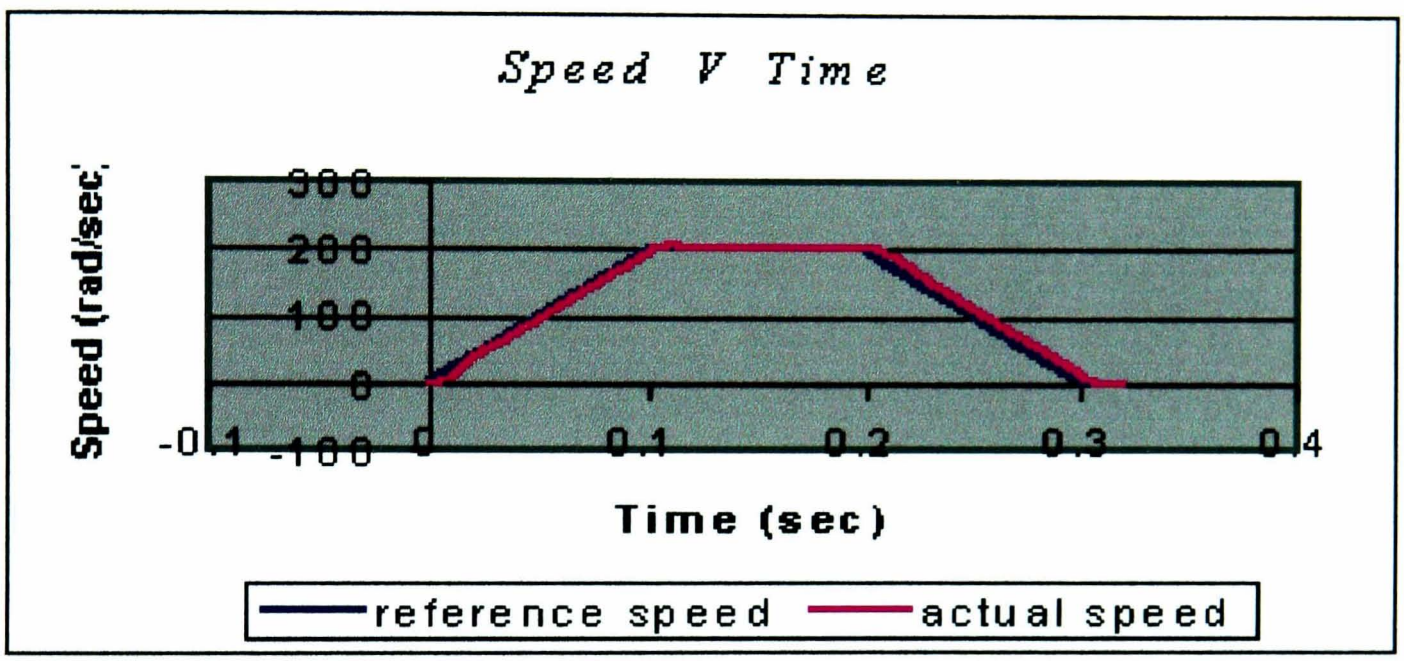


Figure 5.27 - *Motor speed and speed reference versus Time*

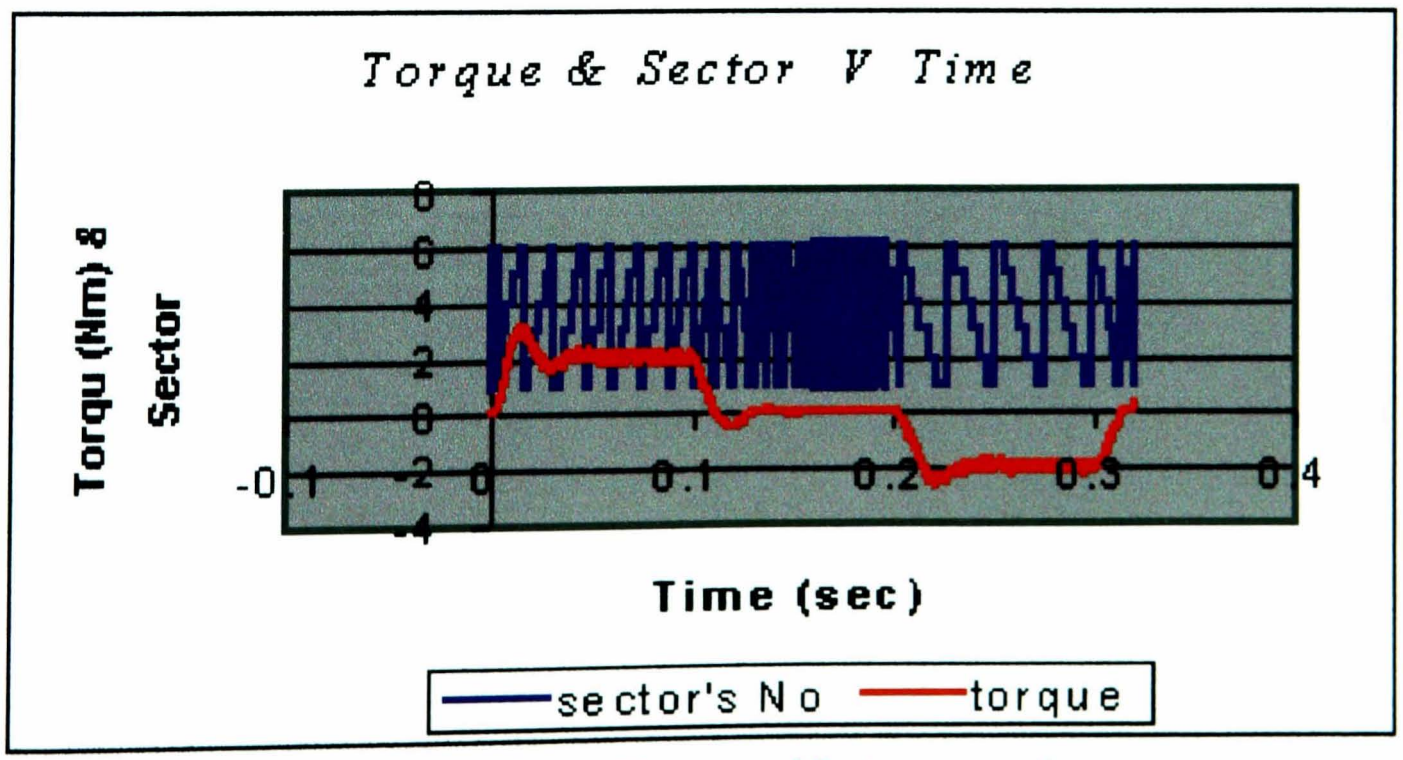


Figure 5. 28 - *Torque and Sector versus time*

Figure 5.27 shows good tracking of the actual speed to the reference speed while Figure 5. 28 shows the expected torque and sector response.

The result in Figure 5.29 to Figure 5.31 illustrates the response of the motor model to the simulation of a step load. The load torque T_l imposed is initially 0 *Nm* and is then stepped to 3 *Nm* at t=0.015 sec.

The time/torque graph of Figure 5.29 shows that the motor torque follows the reference load torque. When simulation time is equal to 0.15 sec, the load step causes a new transient period and after the oscillations have diminished, the torque has reached 3 *Nm*. From Figure 5.30 the speed is seen to decrease from 150 rad/sec to almost 140 rad/sec when steady state is reached for the second time. This shows that the motor model is capable of tracking and following the load torque demand. However, as the supply voltage has been kept constant during the simulation time, an increase in the required level of torque causes the motor to lose speed. This can be seen in Figure 5.30 where the speed/time graph is plotted, again proving that the drive model behaves correctly and the response is stable.

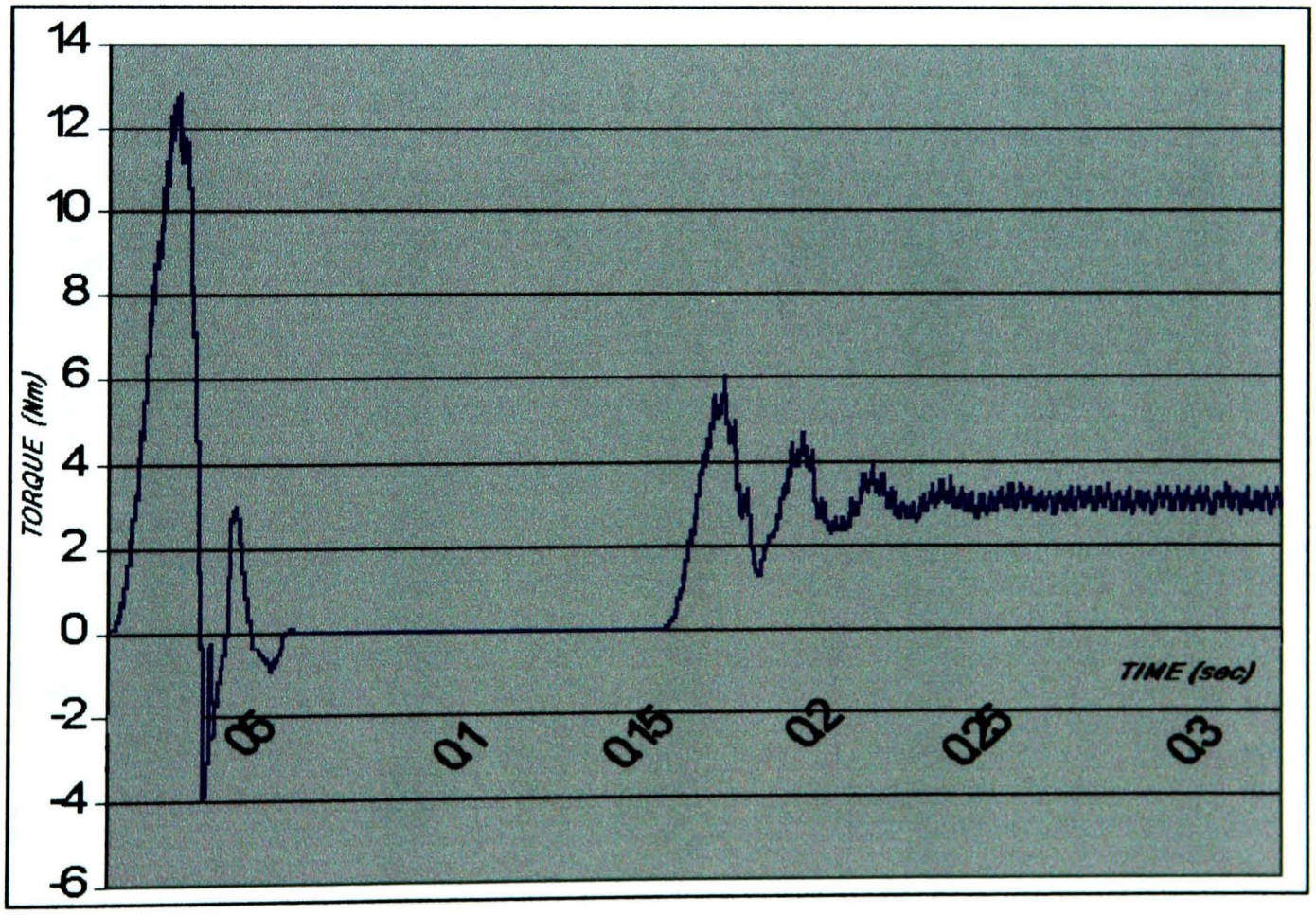


Figure 5.29 - *Torque /Time graph T_l = 3 Nm, J_r=0.001 Kg.m^2*

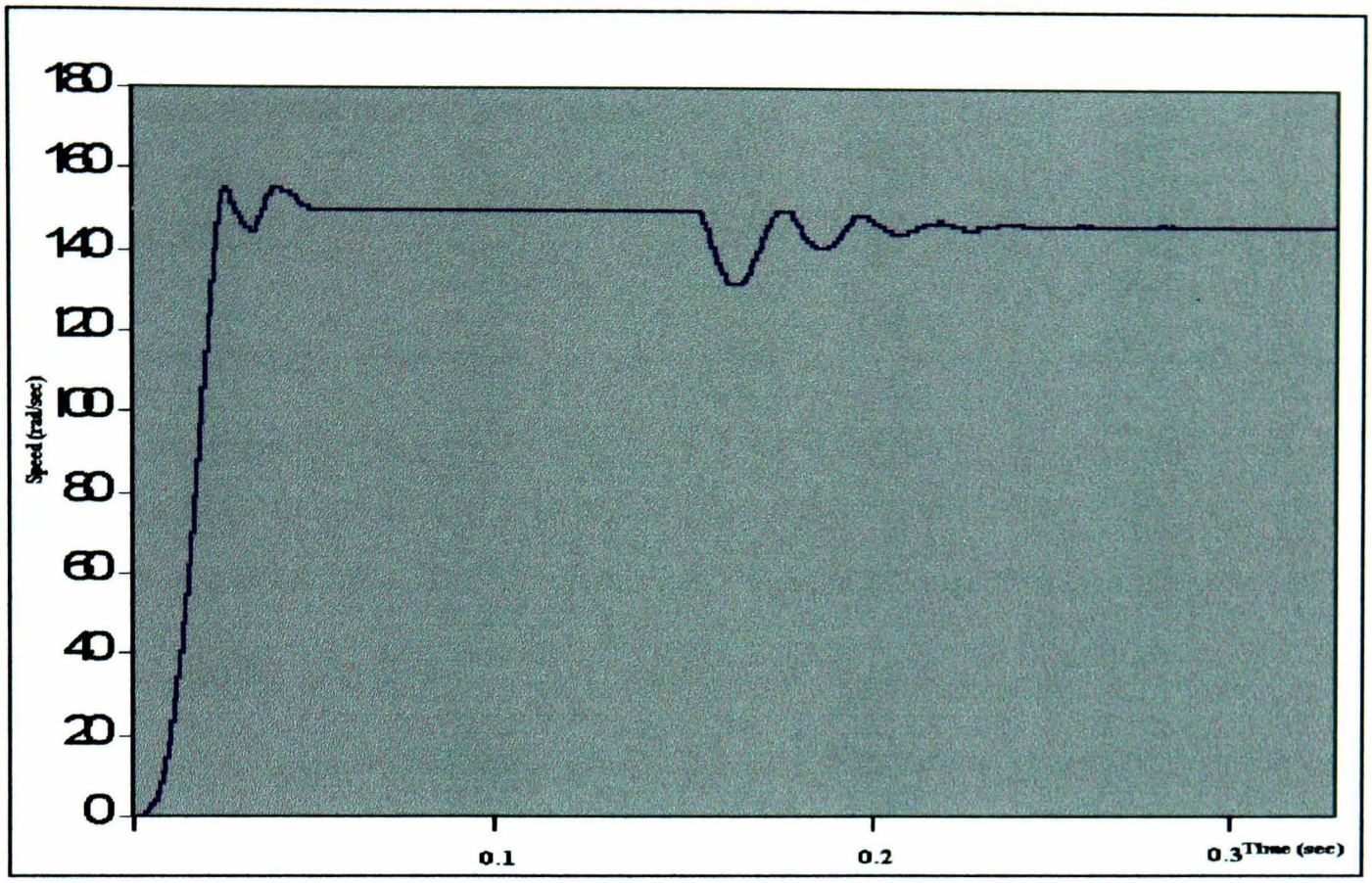


Figure 5.30 - *Speed/Time graph T_L= 3 Nm, J_r= 0.001 Kg. m^2*

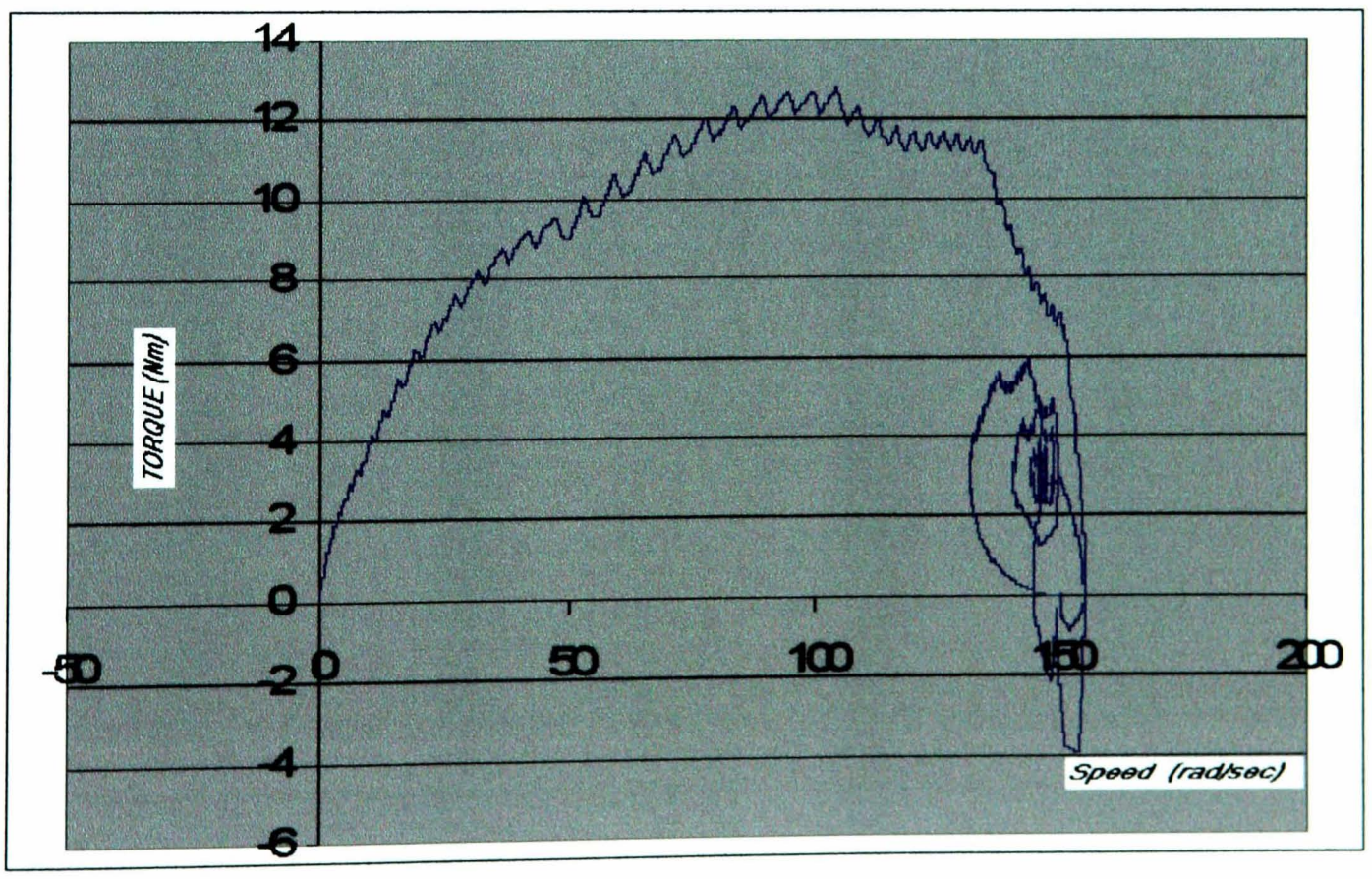


Figure 5.31 - *Speed/Torque graph T_l = 3 Nm, J_r=.001Kg.m^2*

Immediately following the start of the drive, the torque reaches a maximum of 13 *Nm* and then decays towards zero. From the equation $T_e = J\frac{d\omega_r}{dt} + D\omega_r + T_l$ and knowing that T_L=3 *Nm* and setting *D=0,* then $J\frac{d\omega}{dt} = T_e - T_L$. From this it is seen that, when the torque developed becomes smaller than the load torque then $J\frac{d\omega}{dt}$ is negative and speed decrease. This is illustrated in Figure 5.31. The torque then increases again and speed recovers in the same manner. In the region 140-155 *rad/sec*, the rotor speed oscillates between under synchoronous and super synchronous speed values before stabilising. Each time the torque becomes smaller than 3 Nm, $J\frac{d\omega}{dt}$ is negative and the speed decreases.

Figure 5.32 to Figure 5.34 illustrate the response of the motor model to the simulation of a load step T_L which is equal to 1 *Nm* at start and then will step up to 3 *Nm*.

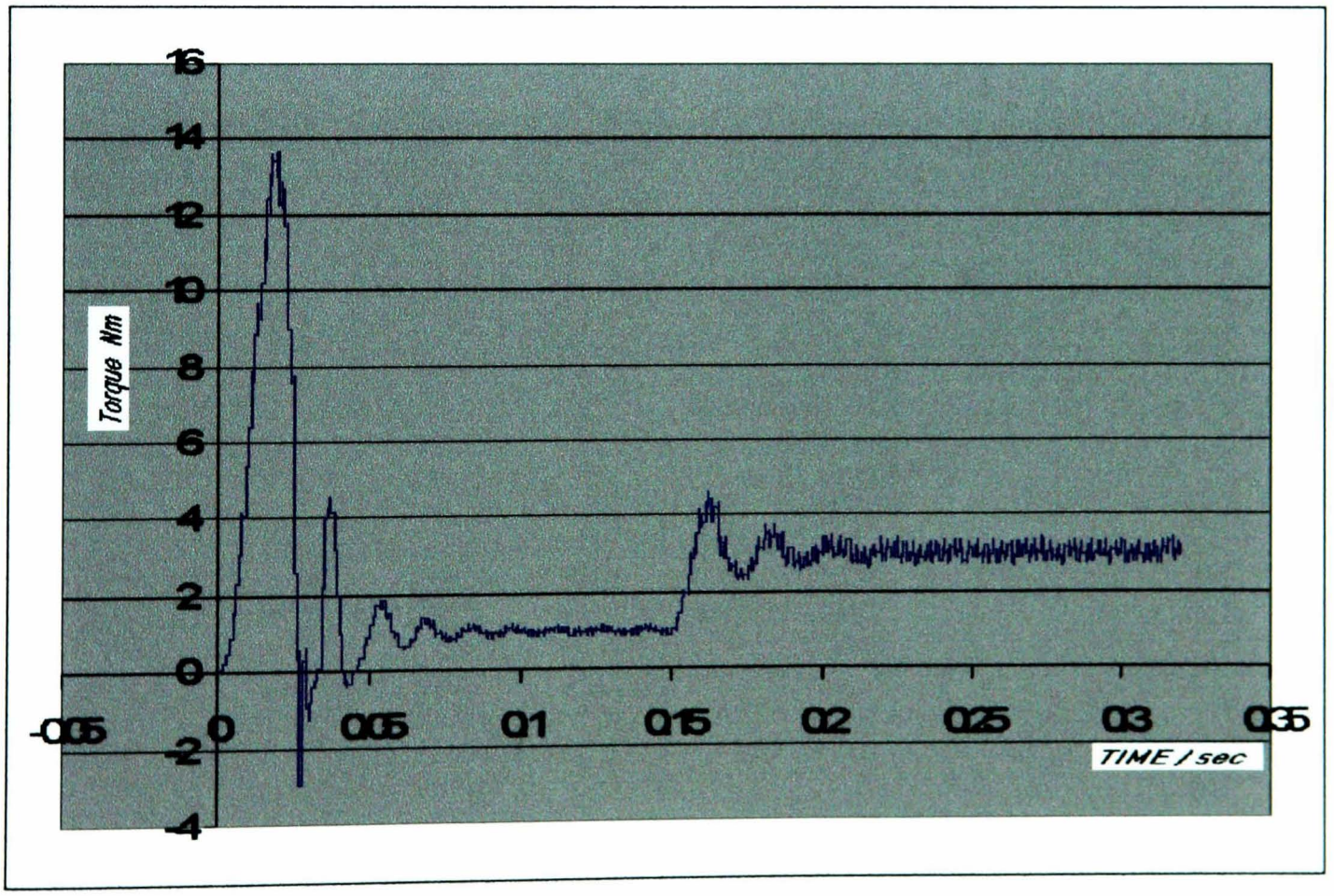


Figure 5.32 - *Torque /Time graph* T_{Lstart} *= 1 Nm,* J_r *= 0.001 Kg.m*^{2}

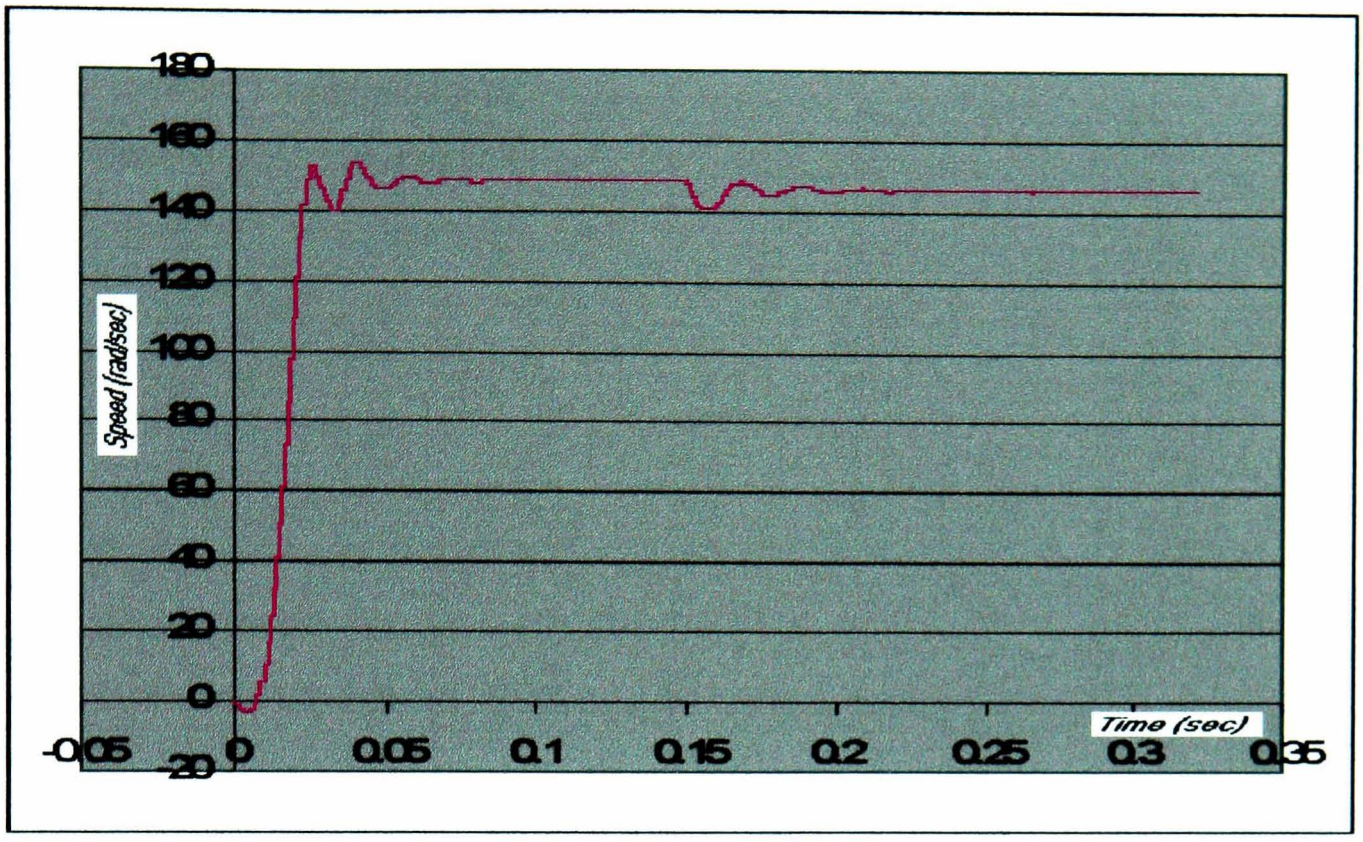


Figure 5. 33 - *Speed /Time graph $T_l = 3$ Nm, $J_r=0.001$ Kg.m^2*

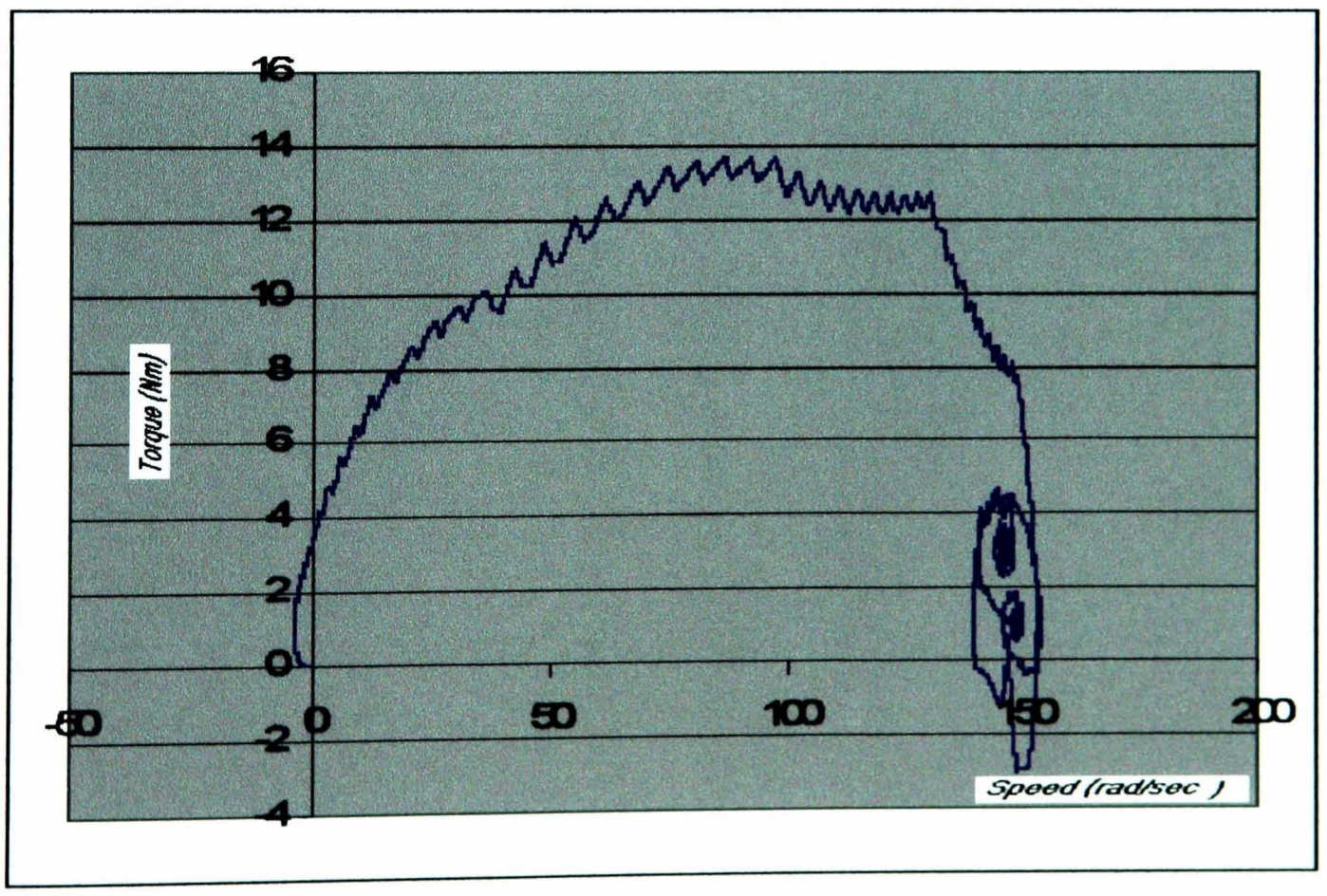


Figure 5.34 - *Speed/Torque graph $T_{l\,start}$= 1 Nm, J_r=.001Kg.m^2*

This chapter presented a new approach to the modelling, simulation and controller design of a complete induction motor electric drive system. The novel technique uses a hardware description language (VHDL) as a unique EDA environment for system modelling, evaluation and controller design. Simulation results are presented, proving the validity of the model for the vector controlled induction motor system. The next chapter proposes a new FPGA based control structure for a.c. drives and develops a novel digital circuit for induction motor vector control, employing a single FPGA Sparttan XCS40 from Xilinx, Inc. An important aspect in modelling the control system is the representation of the machine using the Clark and Park transforms, theta calculation, multiplication, division, PID controller and PWM. Their representation in VHDL is described, followed by implementation in FPGA. The PWM waveform generator and performance characteristics are then demonstrated and the overall strategy is validated by the results obtained.

Chapter 6

VHDL Design of the Field Oriented Controller (FOC)

The design of the FOC presented in chapter 4 did not take into account the synthesis and implementation considerations or the limitations of the target technology. These consideration have to be addressed when preparing a design for downloading into hardware, usually a PLD, FPGA or ASIC. Using the Xilinx Foundation Series F1.5 EDA tool, a VHDL design can be synthesised then implemented for a particular device. Before a VHDL design implementation can proceed, all the VHDL functions not synthesisable have to be eliminated and replaced with functions that can be translated into hardware. A function is considered not to be synthesisable if the synthesis tool can not readily convert it into a hardware description. For example functions such as mathematical division, multiplication and trigonometric operations are not synthesisable using currently available synthesis tools.

The approach adopted is to break down the digital controller into components where each component performs a specific function such as a Clark transform, Park transform, multiplication, division, theta calculation, PID controller and PWM. The overall block diagram of the FOC is illustrated in Figure 6.1 and the realization of the elements is described in this chapter.

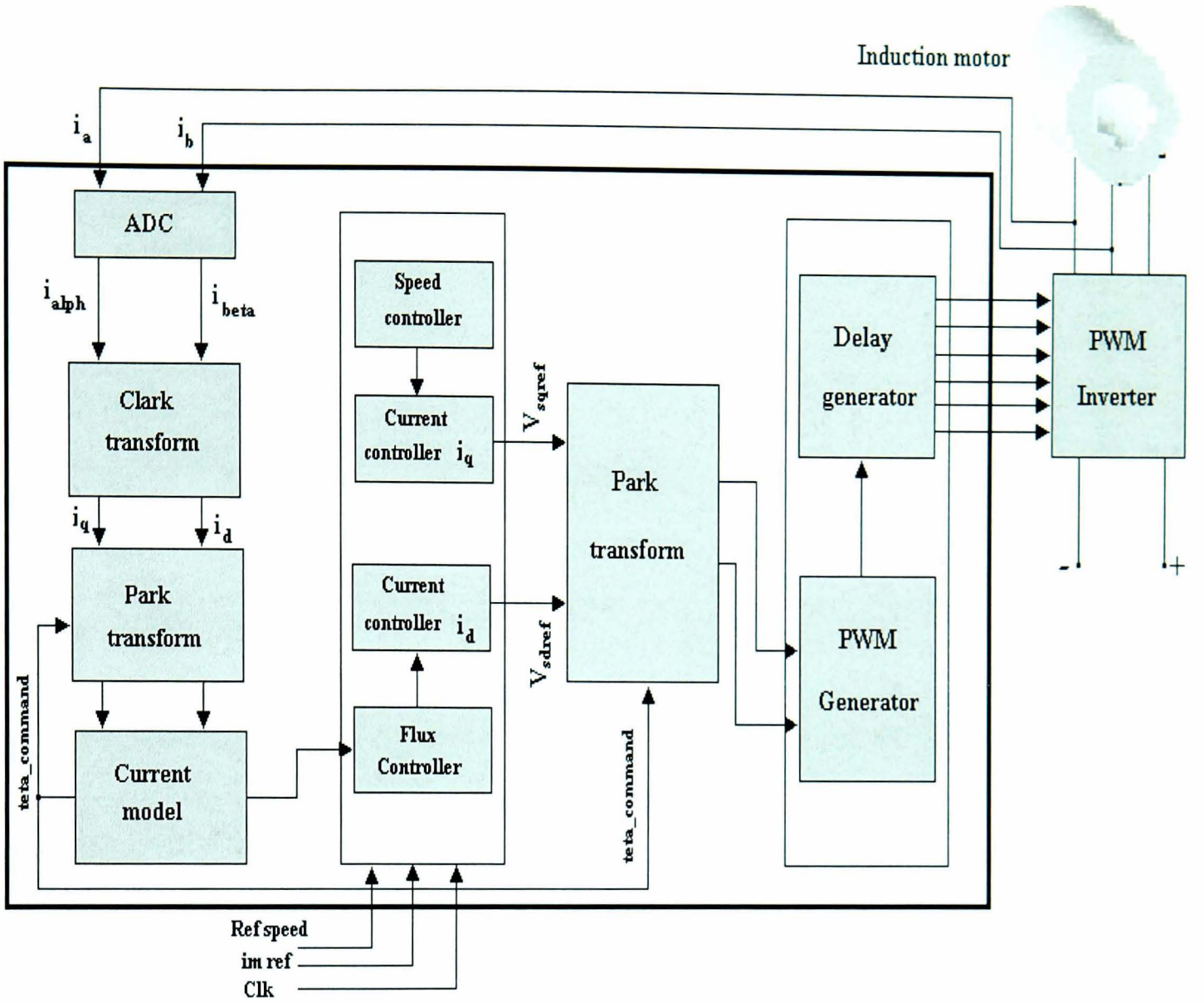


Figure 6.1 - *The overall block diagram of the FOC*

6.1 The 3 Axis – 2 Axis Currents Transformation in VHDL

The Clark transform uses three phase currents i_a, i_b and i_c to calculate the currents in the two phase orthogonal stator reference frame i_α and i_β [66], [67]. Figure 6.2 shows a block diagram which represents the mathematical transformation of a three phase system into a two phase orthogonal system.

Considering a particular case with i_α superimposed with i_a and $i_a + i_b + i_c = 0$, i_a, i_b and i_c can be transformed to i_α and i_β by the mathematical transformation described below [68].

For a 3 phase balanced load: $i_a + i_b + i_c = 0$ therefore $i_a = -(i_b + i_c)$.

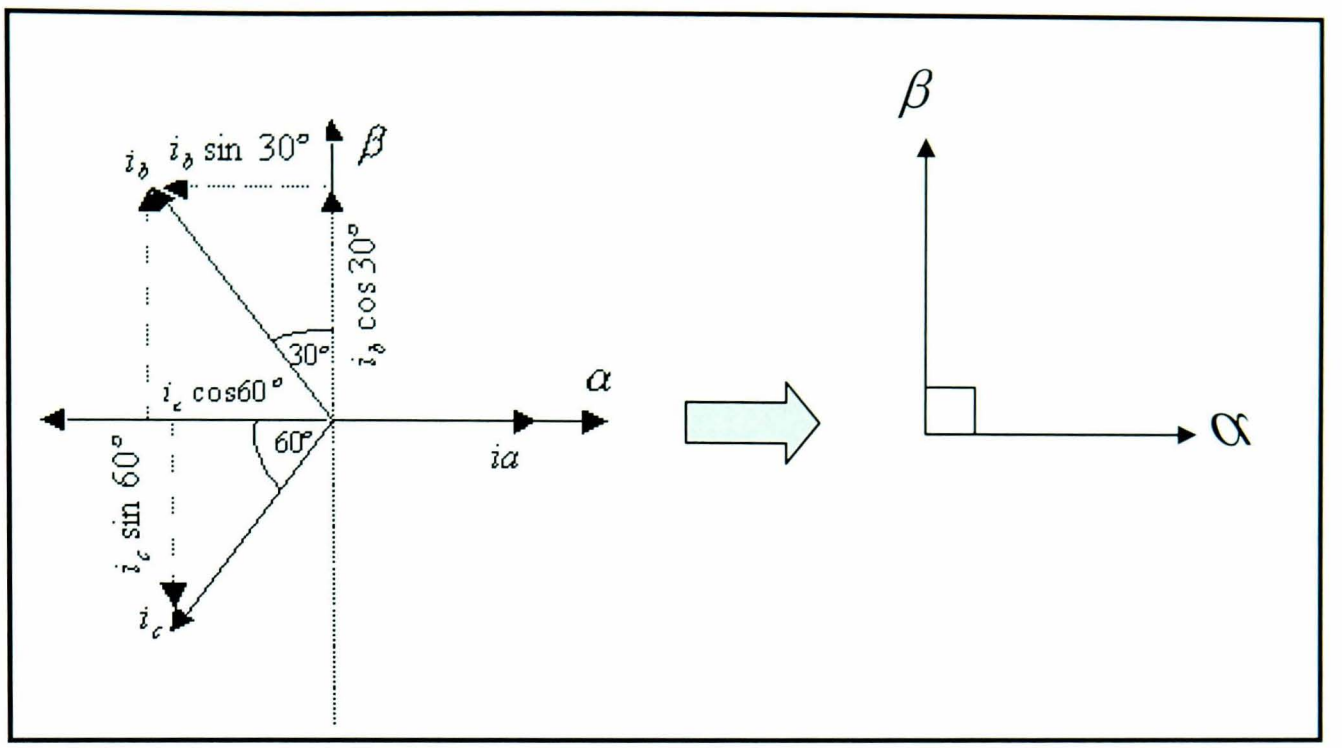


Figure 6.2 - *3-2 Current transformation*

From Figure 6.2, i_α and i_β can be found as :

$$i_\alpha = i_a - i_b \sin 30^o - i_c \cos 60^o$$

$$i_\alpha = i_a - 0.5 i_b - 0.5 i_c$$

$$i_\alpha = i_a - \frac{1}{2}(i_b + i_c)$$

$$i_\alpha = i_a + \frac{1}{2} i_a$$

$$i_\alpha = \frac{3}{2} i_a \qquad \textbf{6-1}$$

$$i_\beta = i_b \cos 30^o - i_c \sin 60^o$$

$$i_\beta = \frac{\sqrt{3}}{2} i_b - \frac{\sqrt{3}}{2} i_c$$

$$i_\beta = \frac{\sqrt{3}}{2}(i_b - i_c) \qquad \textbf{6-2}$$

If equations **6-1** and **6-2** are multiplied by $\frac{2}{3}$ then

Equation **6-1** becomes $i_\alpha = i_a$ **6-3**

Equation **6-2** becomes $$i_\beta = \frac{1}{\sqrt{3}}(2i_b + i_a) \qquad \textbf{6-4}$$

In order to solve equation **6-4**, an integer approximation of the $1/\sqrt{3}$ is expressed as:

$\frac{1}{\sqrt{3}} = 0.577 = 0.100100111011$ in binary.

Digital representation in 2 complementary of 0.577 using VHDL can be achieved by considering either the 1's or the 0's. If 1's are considered, then 0.577 can be approximated by:

$$\frac{1}{\sqrt{3}} = 0.577 = \frac{1}{2^1} + \frac{1}{2^4} + \frac{1}{2^7} + \frac{1}{2^8} + \frac{1}{2^9} + \frac{1}{2^{11}} + \frac{1}{2^{12}}$$

$$\approx 0.5+0.0625+0.00781+0.0039+0.0019+0.000488+0.000244$$

$$\approx \underline{\mathbf{0.5769}}$$

If 0's are considered, then 0.577 can be approximated by:

$$\frac{1}{\sqrt{3}} = 0.577 \approx 1 - \left(\frac{1}{2^2} + \frac{1}{2^3} + \frac{1}{2^5} + \frac{1}{2^6} + \frac{1}{2^{10}} \right)$$

$$\approx 1 - (0.25+0.125+0.03125+0.015625+0.000976)$$

$$\approx \underline{\mathbf{0.57714}}$$

The second method has been chosen and is represented in VHDL as shown in Figure 6.3.

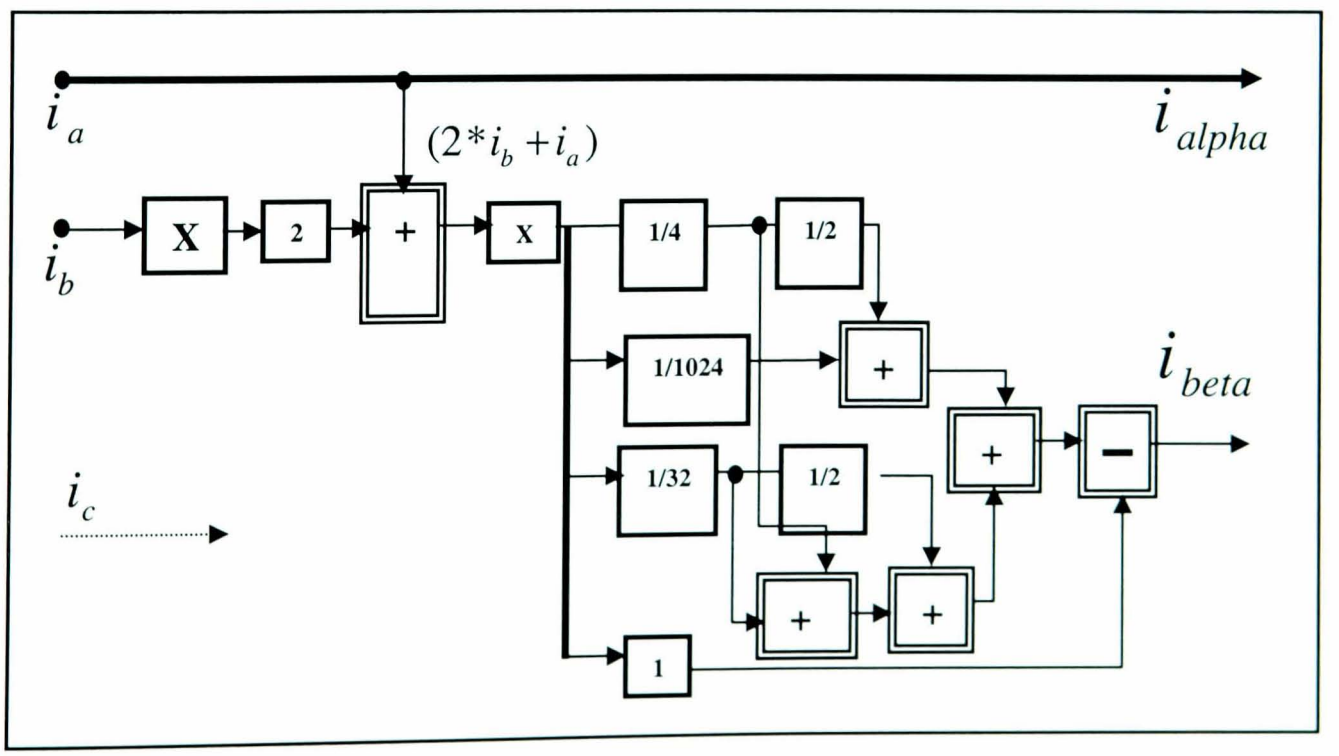


Figure 6.3 - *The 10 bit circuit realization of the 3 phase-2 axis current converter*

The ENTITY "Clark transform" declaration is straightforward, simply specifying the input signals (i_a and i_b) and output signals (i_{alpha} and i_{beta}), together with their data types (in this case, 10-bit Std_Logic_Vector).

```
Entity clrktransform  is
                port (
                ia,ib: in STD_LOGIC_VECTOR (8 downto 0);
                ialpha: out STD_LOGIC_VECTOR(9 downto 0 );
                ibeta: out STD_LOGIC_VECTOR(9 downto 0)
                  );
End clrktransform;
```

However there are a number of ways to describe the required behaviour of the above equations in VHDL (similarly, there are usually many different ways in which an algorithm can be expressed in any given programming language). Equation **6-3** is easily implemented in VHDL, while equation **6-4** is realised as a purely combinational circuit using a single PROCESS statement called Process (ibeta_par). The arithmetic operations required to implement this method and to achieve the scaling of data values by factors of 2 and $\frac{1}{\sqrt{3}}$ are: Scale a binary value by 2 by shifting the value one bit left (thus discarding the most significant bit). Scale a value by $\frac{1}{\sqrt{3}}$, without using multiplication or division, requires that the schematic shown in Figure 6.3 is implemented. Part of the Clark transform VHDL code is shown below.

```
Begin
  ialpha<=(ia(8) & ia);
  ibeta_par<=(ib & '0') + ia;
process(ibeta_par)
  variable rez: std_logic_vector(21 downto 0);
begin
  rez := ibeta_par & zeroes(12);
  rez := rez - (ibeta_par & zeroes(10));
  rez := rez - (ibeta_par & zeroes(9));
  rez := rez - (ibeta_par & zeroes(7));
  rez := rez - (ibeta_par & zeroes(6));
 rez := rez - (ibeta_par & zeroes(1));
 rez := rez - ibeta_par;
 ibeta<=rez(21 downto 12);
End process;
```

The ADC samples the currents (i_a and i_b) and converts them to a nine-bit signed binary number. The output of the ADC is used as an input to the Clark transform program written in VHDL as shown in Figure 6.4.

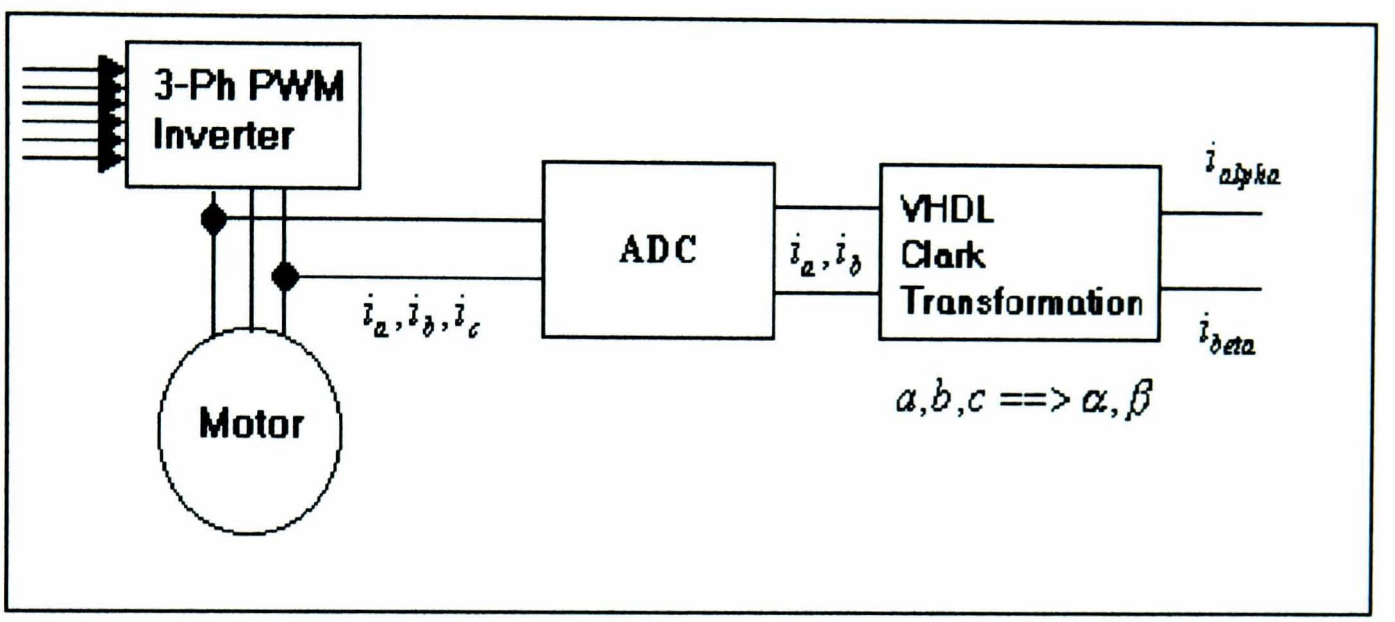


Figure 6.4 - *Block diagram of ADC and Clark transformation*

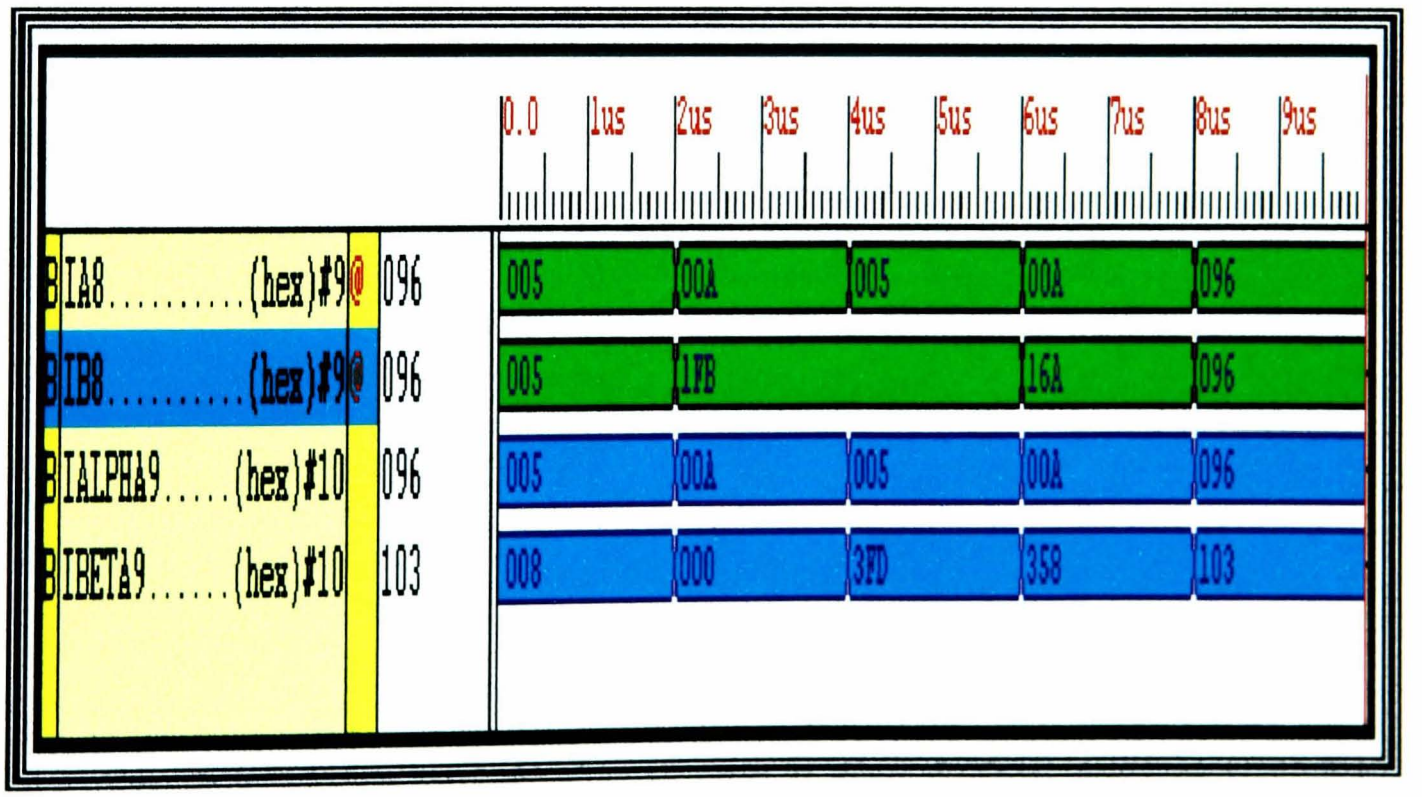


Figure 6.5 - *Simulated waveforms for 3-2 Clark transformation*

The VHDL description can be simulated to confirm the required behaviour. Figure 6.5 shows a screen copy of the simulation environment used to test the VHDL code. In order to check the result, the last value of Figure 6.5 was taken as an example:

$i_a = 96\ Hex = 150\ Decimal\ in\ (Amperes).$

$i_b = 96\ Hex = 150\ Decimal\ in\ (Amperes).$

$$i_{beta} = \frac{1}{\sqrt{3}} = (2 * i_b + i_a)$$

$$i_{beta} = \frac{1}{\sqrt{3}} * (2*150+150) = 259\,Decimal$$

$= \boldsymbol{103\ hexadecimal.}$

The view of the VHDL source code as well as the simulation results gives the engineer further debugging capabilities such as the setting of breakpoints in the code, single-stepping through the VHDL and examining variables as necessary.

Once the VHDL code is synthesised and simulated, the next step is to implement the design using the implement design window as shown in Figure 6.6. The flow engine windows shows a graphical rendition of the stages required to map a netlist into a FPGA (in this case the target is Xilinx XC4010E).

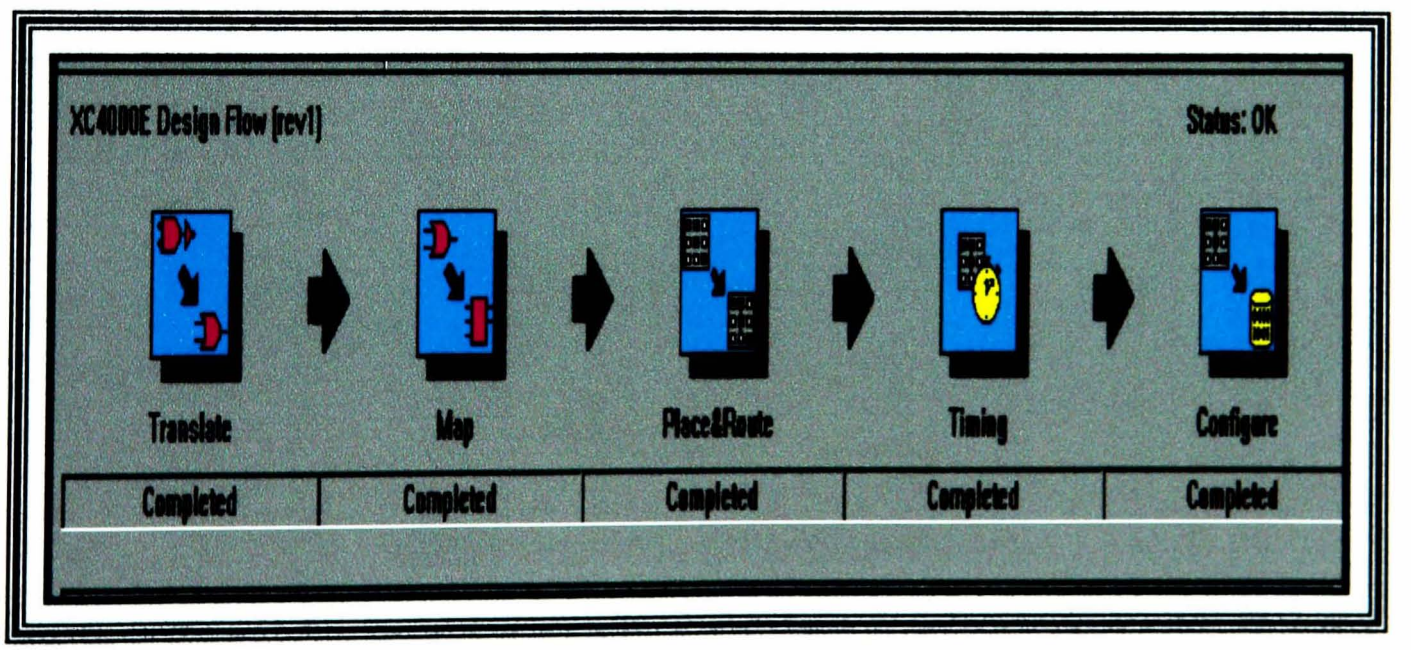


Figure 6.6 - *Flow engine windows (The processing steps for mapping a netlist into FPGA)*

A partial layout of the Clark transform synthesised into an FPGA target is shown in Figure 6.7. This shows the wiring of FPGA cells for the implementation of the Clark transform into a Spartan S40PQ208 Xilinx chip.

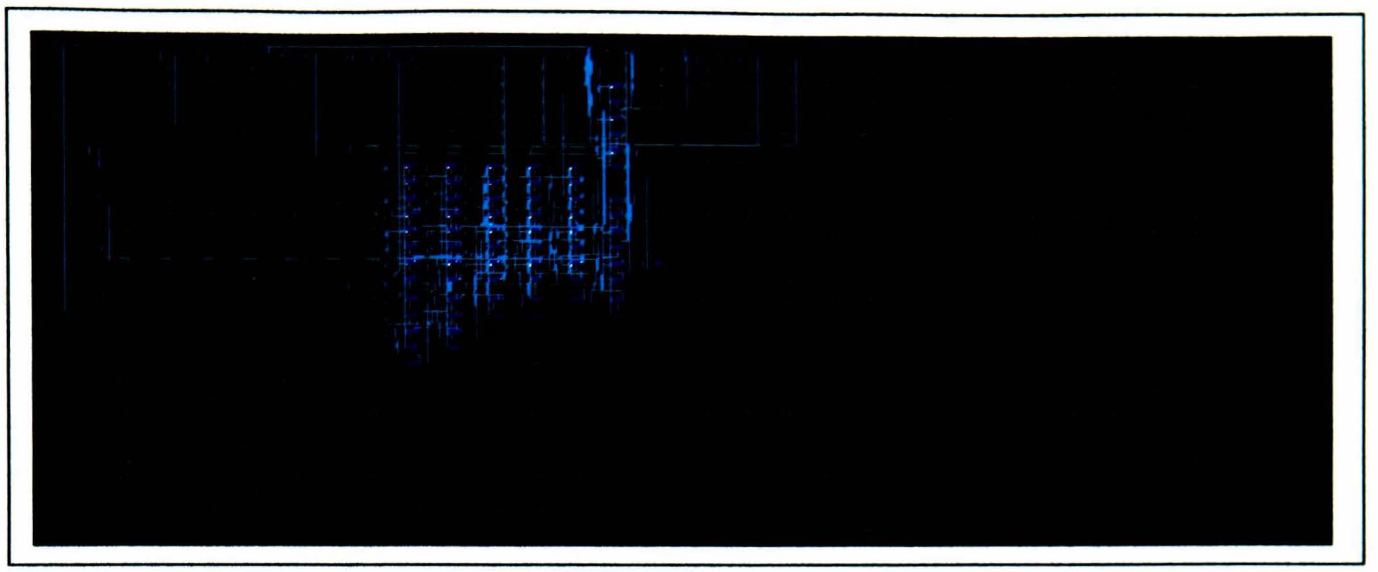


Figure 6.7 - *FPGA layout of Clark transform*

6.2 The (α, β) to (d, q) Projection (Park transformation)

Park transformation is the most important transformation in the FOC. The two phases (α, β) frame representation calculated within the Clark transform is fed to a vector rotation block where it is rotated through an angle ϑ to follow the d, q frame attached to the rotor flux. If the d-axis is considered to be aligned with the rotor flux, then Figure 6.8 shows the relationship between the two reference frames for the current vector.

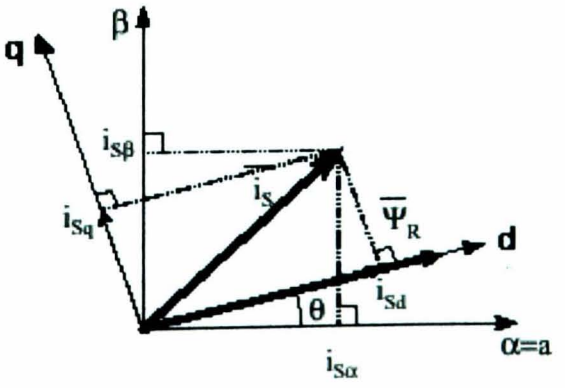


Figure 6.8 - *Stator current space vector*

The rotation through an angle ϑ is done according to the formulae:

$$i_{sd} = i_{alpha} * \cos(teta) + i_{beta} * \sin(teta) \qquad \textbf{6-5}$$

$$i_{sq} = -i_{alpha} * \sin(teta) + i_{beta} * \cos(teta) \qquad \textbf{6-6}$$

Before a VHDL design can be synthesised all VHDL functions which are not synthesisable have to be eliminated and replaced with functions that can be translated into hardware. It is obvious from equations **6-5** and **6-6** that the Park transformation requires several multiplications, additions and subtractions.

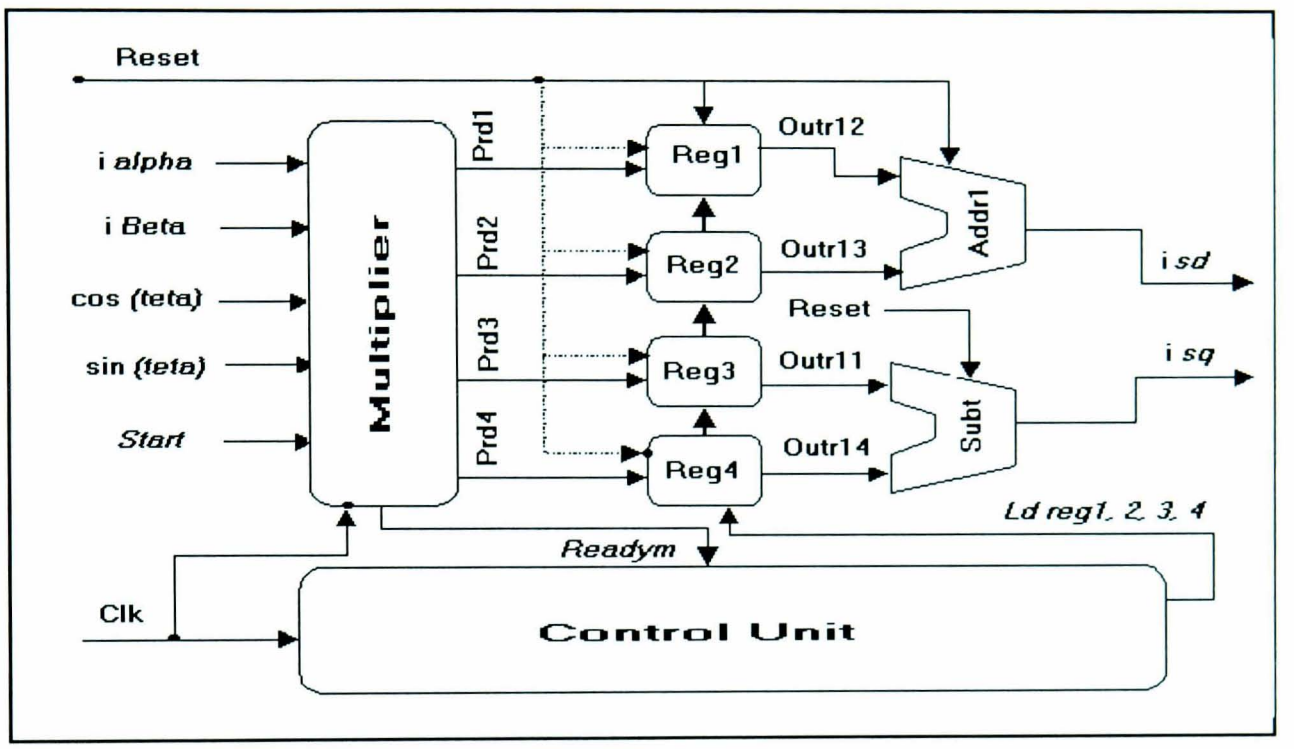


Figure 6.9 - *Block diagram of Park transformation*

The VHDL description of Equations **6-5** and **6-6** is represented in Figure 6.9 as a block diagram. Figure 6.9 is divided into 8 VHDL components. Each component is depicted with its VHDL code file '<filename>'.vhd'. The VHDL structure of the Park transformation comprises:

- ***Control unit .***
- ***Multiplier .***
- ***Register1, Register2, Register 3 and Register4 .***
- ***Adder and Subtract .***

6.2.1 Control Unit Description

The interface description of the control unit corresponding to its logical symbol is shown in Figure 6.10.

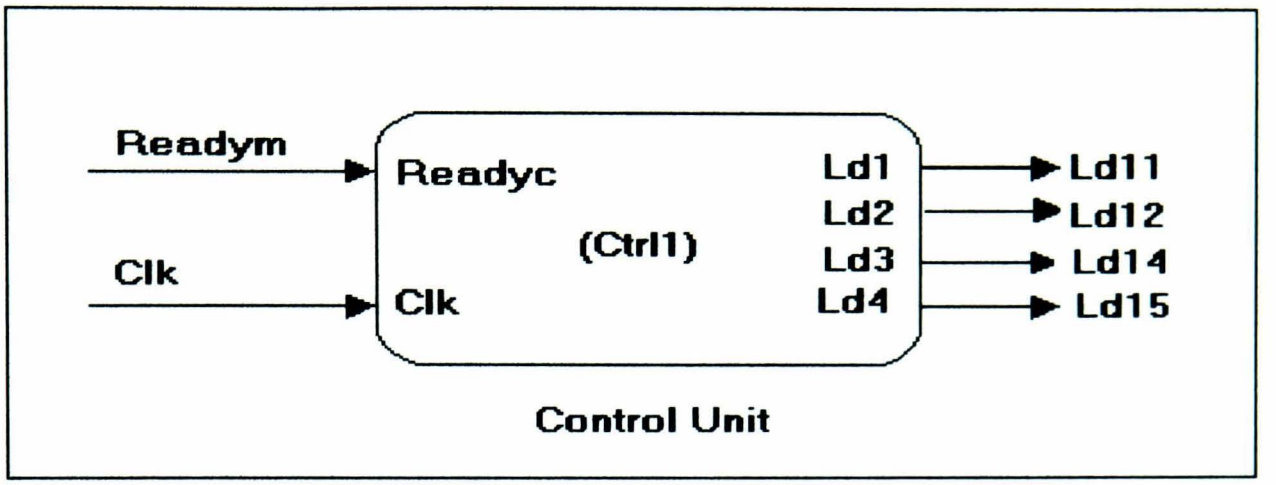


Figure 6.10 - *Logical symbol of the Control unit*

Part of the VHDL code for the control unit is shown (the complete code is included in Appendix B). The entity declaration contains five input ports and four output ports.

```
ENTITY ctrl IS
 PORT( clk,readyc,readyc1,readyc2,readyc3 : in std_logic;
       ld1:out std_logic;
          ld2:out std_logic;
                ld3:out std_logic;
                ld4:out std_logic);
END ctrl;
```

Within the control unit the interfacing components, multiplier, registers, adders and subtract are instantiated and connected to one another. The unit also features two clocking signals for synchronising the internal components. Part of the functionality of Ctrl1 is described by an architecture identified as Ctrl1_arch as shown:

```
Architecture ctrl_arch of ctrl is
begin
 process (clk,readyc,readyc1,readyc2,readyc3)
  begin
  --**********************
         if (clk='1' and clk'event ) then
          if readyc='1' then
           ld1<='1';

           else
```

```
        ld1<='0';
        end if;
      ... ... ... ... ... ... .
  end process;
End ctr1_arch;
```

The process statement has an if statement with a clk = '1' AND clk'EVENT condition that encloses all the assignments to the Ctrl1 output. This condition becomes true when an event occurs on the binary signal clk that causes the value of this signal to become '1'. In other words, the condition becomes true when the rising edge of the clk is observed. On this edge, if readyc='1' , Ld1 becomes '1'; otherwise '0' will be assigned to Ld1. The procedure is also used for Ld2,Ld3 and Ld4.

6.2.2 The Multiplier

Figure 6.11 shows the decomposition of the multiplication of operands A and B into a series of simpler calculation cycles. Each operation cycle consists of multiplying the operand A with the least significant bits of B and adding the result into a shift register. Both the result and operand B are then shifted with the LSB positions to the right.

The normal procedure for binary multiplication can be described by the following example, with A= 0011 and B = 0101:

	0011	A = Multiplicand
X	0101	B = Multiplier
=	0011	A
	0000	0
	0011	A shift left by 2 bits
	0000	
=	0001111	Final Result

The multiplication process effectively becomes a combination of an addition and shift left operation.

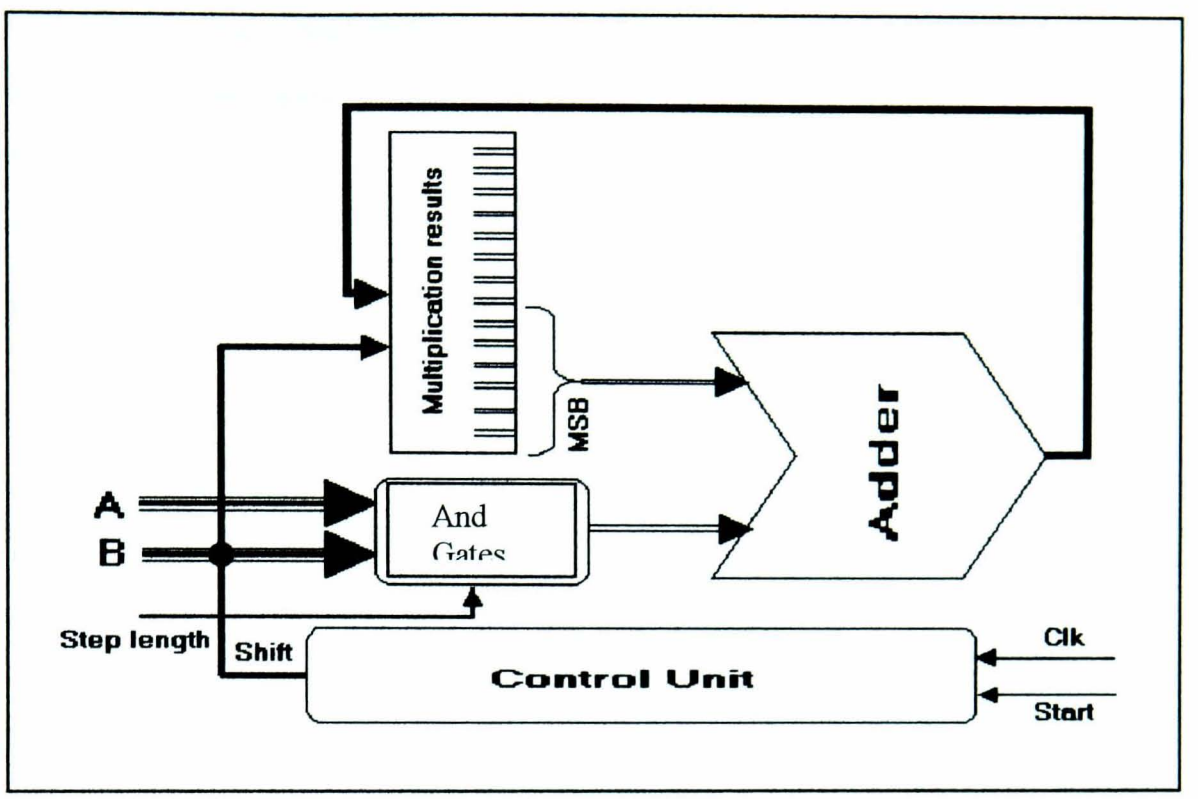


Figure 6.11 - *Block diagram of multiplier structure*

Therefore a VHDL process called Process (a,b.int_prod) was designed. The multiplication result is stored into the signal named INT_PROD that models the multiplier main shift register. Part of the multiplier code is given as:

```
Entity multiplier is
        Generic(
         n: in integer:=10;  -- The operand length.
         m: in integer :=8;  -- The result length.
         step_length:in integer:=1
            );
 port (
        a: in std_logic_vector (n-1 downto 0); -- Can be only positive.
         b: in std_logic_vector (m-1 downto 0); -- Can be both positive and negative.
    prod: out std_logic_vector (n+m-1 downto 0);
            clk,start: in std_logic;
            ready: out std_logic
 );
 End multiplier;
```

The multiplication process is triggered by the START input signal as illustrated by Figure 6.12. When this signal is active, the control unit loads operand B into the corresponding shift register and initialises the result register with zeroes. The internal signal COUNT is loaded with value NSTEPS when the START input signal is activated and is simultaneously decreased at each calculation cycle by adding the partial multiplication result to the main shift register. If COUNT is larger than 1, then the two registers are shifted and a new cycle is initiated, otherwise the calculations are stopped and the READY signal is activated. Once the READY signal is activated it will, in turn, activate the control unit process which is used to activate the Ld1, Ld2, Ld3 and Ld4 registers setting to a value of '1' or '0'.

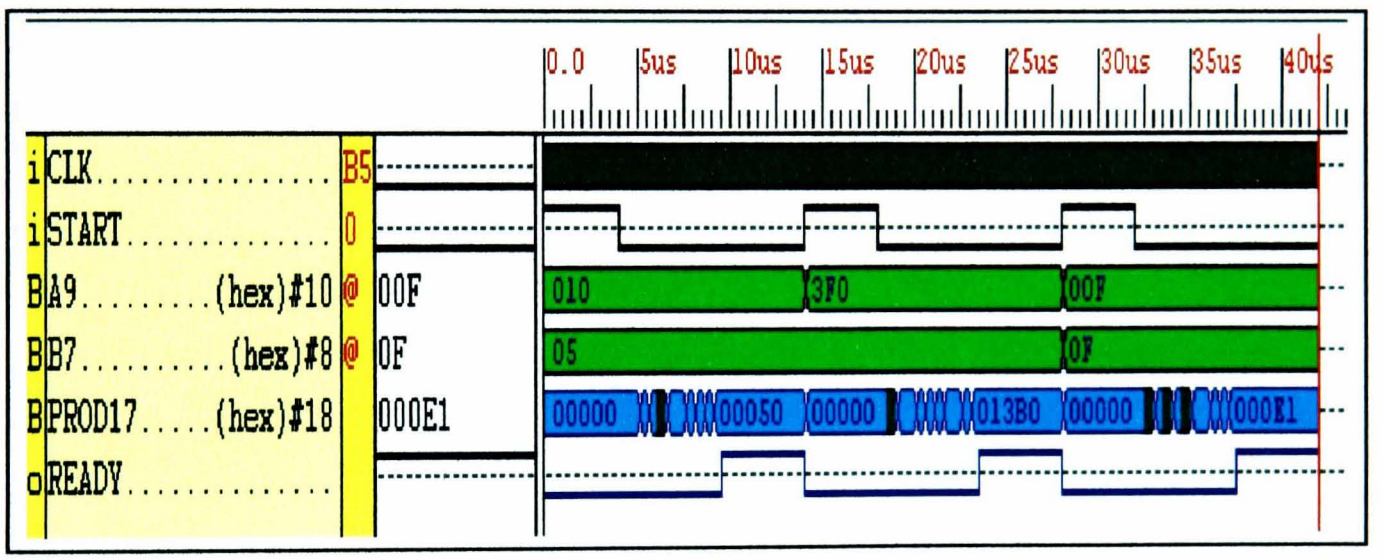


Figure 6.12 - *Simulated waveforms for the multiplier*

VHDL simulation tools provide mechanisms for applying test inputs and observing output waveforms. Figure 6.12 shows the simulation result of a netlist architecture of a multiplier. In order to check the result the values of Figure 6.12 are taken as an example (all values are in hexadecimal):

- **A = 10 and B = 5 therefore PROD = A * B = 10 * 5 = <u>50</u>**
- **A = -10 = 3F0 and B = 5 therefore PROD = A * B = 3F0 * 5 = <u>13B0</u>**
- **A = F and B = F therefore PROD = A * B = F * F = <u>E1</u>**

6.2.3 Registers

Registers are very common components of any digital circuit. Their role is to store strings of bits and in some cases to perform very basic operations like shifting or rotating these bits. Reg1, Reg2, Reg3 and Reg4 in Figure 6.9 are used as a temporary storage for the products $i_{alpha} * \sin(teta)$, $i_{beta} * \cos(teta)$, $i_{alpha} * \cos(teta)$ and $i_{beta} * \sin(teta)$. The algorithm which represents these registers is achieved with the following VHDL statements.

```
Entity reg1 IS
Port( ld11,reset : in std_logic;
      in_reg1: in std_logic_vector(10 downto 0);
      out_reg1: out std_logic_vector(10 downto 0));
End reg1;

Architecture reg1_arch OF reg1 IS
Begin
process (ld11,reset)
  Begin
   If reset ='1' then
        out_reg1<=(others=>'0');     -- to intialize register q and q1 to 0.
   Elsif  (ld11='1'and ld11'event ) then
     out_reg1 <= in_reg1;
   End if;
 End process;
END reg1_arch;
```

The outputs of Reg1, Reg2, Reg3 and Reg4 are fed as inputs to the adder and subtracter in Figure 6.12. The add and subtract functions are achieved using 'STD_LOGIC_SIGNED' packages which are part of the IEEE library. These packages overload the basic arithmetic operators and define the corresponding operations for STD_LOGIC_VECTOR data type as shown by the following VHDL code:

```
Entity addr IS
Port ( reset:in std_logic;
 out_reg11,out_reg12,out_reg13,out_reg14: in std_logic_vector (10 downto 0);
 ids:out std_logic_vector(11 downto 0);
 iqs:out std_logic_vector(11 downto 0));
End addr;
Architecture addr_arch OF addr IS
Begin
```

```
Process (reset,out_reg11,out_reg12,out_reg13,out_reg14)
        Begin
          If reset ='1' Then
             ids<=(others=>'0');
             iqs<=(others=>'0');
          Else
             ids <= out_reg12+out_reg13;
             iqs <= -out_reg11+out_reg14;
 End if;
 End process;
End addr_arch;
```

The '+' operator always generates a resulting std_logic_vector signal or variable whose length equals the maximum length between the two inputs. As inputs out_reg11 , out_reg12, out_reg13 , out_reg14 are 11 bits long, the result would be 11 bits long as well. For the adder to operate correctly and to avoid an overflow it is necessary that the result is 12 bits long.

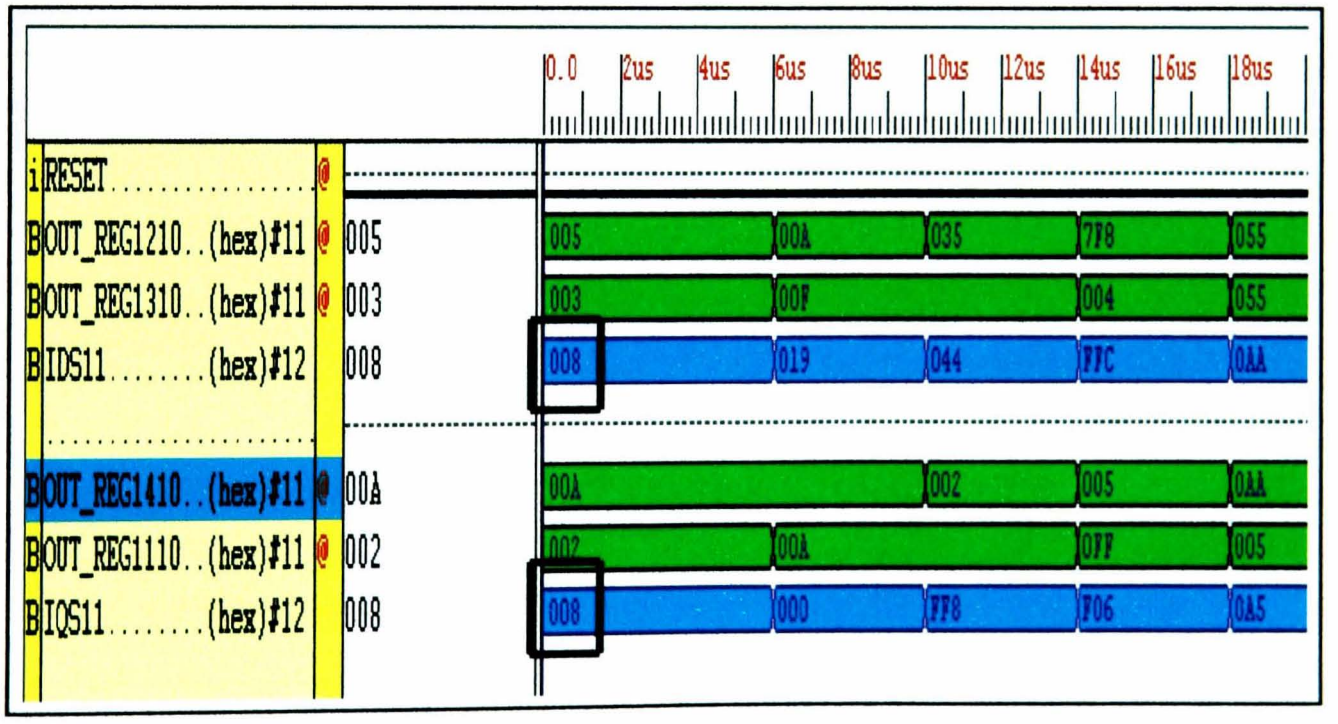


Figure 6.13 - *Simulated waveforms for the adder and subtractor*

Figure 6.13 shows a simulation run result of the netlist architecture of the adder and the subtractor. In order to check the result some values from Figure 6.13 are taken as examples (all values are in hexadecimal):

$$Out\_reg11 = 2 \qquad Out\_reg12 = 5 \qquad Out\_reg13 = 3 \qquad Out\_reg14 = A$$

$$ids = out\_reg12 + out\_reg13 = 5 + 3 = \underline{8}$$

$$iqs = -out\_reg11 + out\_reg14 = -2 + A = \underline{8}$$

Following the synthesis of the gate-level circuit and before implementation, the design may be simulated again to verify that the required performance targets have been achieved. As an illustration some results are shown in Figure 6.14.

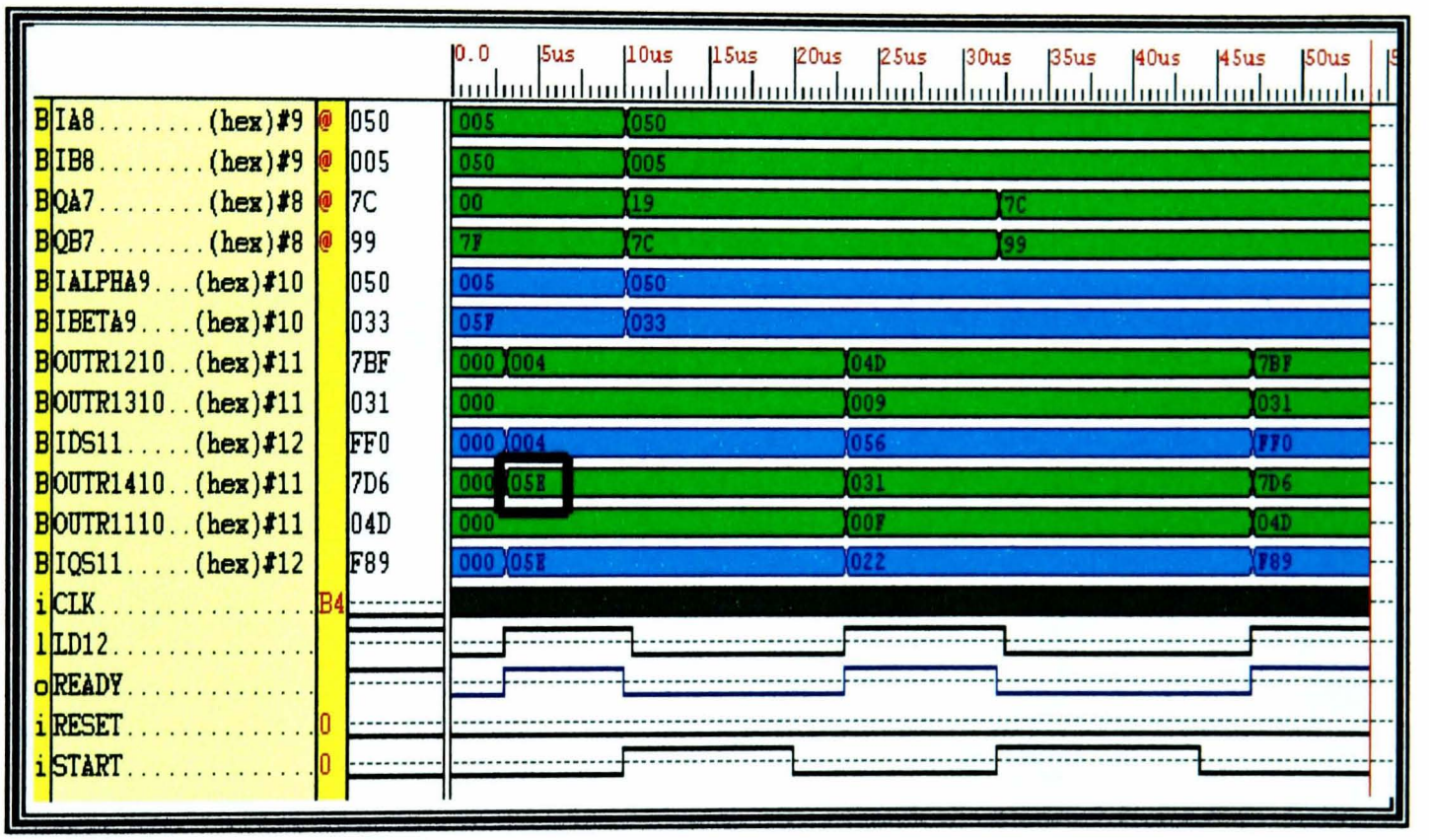


Figure 6.14 - *The over all simulated waveforms for Park transform*

Values were taken as an example to verify the simulation results shown in Figure 6.14 for the digital circuit achieving the Park transformation structure in Figure 6.14. Note that QA and QB represent sin (theta) and cos (theta) respectivelly.

If isalpha =5, isbeta = 5F, qa=0 and qb =7F

Then

$$OUTR11 = isalpha * sin(theta) = 5 * 0 = \underline{0}$$

$$OUTR12 = isalpha * cos(theta) = 5 * 7F = \underline{27B}$$

$$OUTR13 = isbeta * sin(theta) = 5F * 0 = \underline{0}$$

$$OUTR14 = isbeta * cos(theta) = 5F * 7F = \underline{2F21}$$

The multiplication results of OUTR11, OUTR12, OUTR13 and OUTR14 are divided by 7F to get the required digital form of each multiplication. This is equivalent to shifting the result by 7 bits to the right and considering only the most significant bits (MSB) of the multiplication result. If OUT14 is taken as an example then

OUT14 = 2F21, can be represented in binary as = **0000101111100100001**.

If the MSBs are considered then the result becomes: **00001011110**. This binary number is equal to **05E** in Hexadecimal, which is identified as a square in Figure 6.14. A partial layout of the Park transform synthesised to an FPGA target is shown in Figure 6.15. The figure shows the wiring of FPGA cells for the implementation of Park transform into a Spartan S40PQ208 Xilinx Chip.

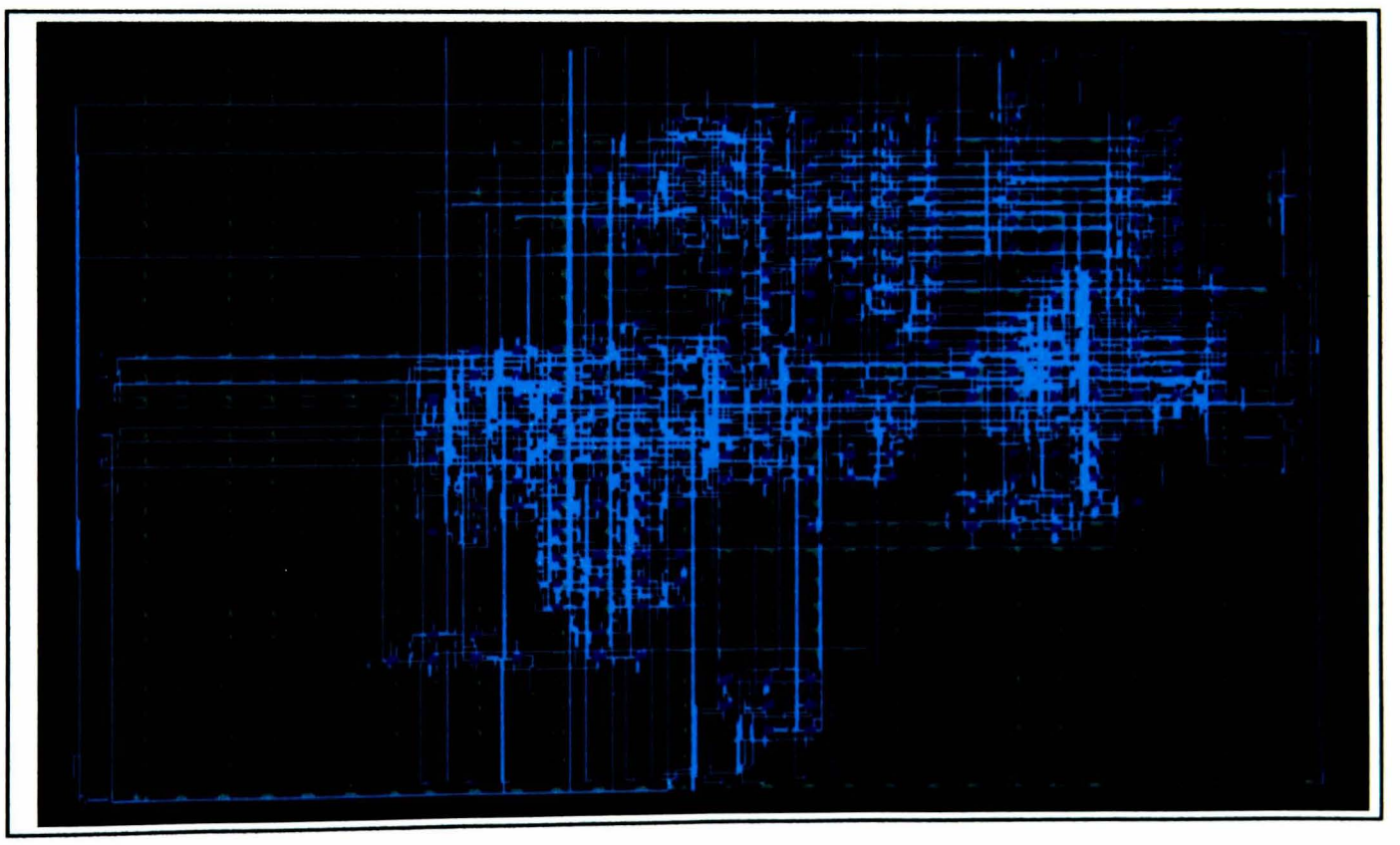


Figure 6.15 - *FPGA layout of Park transformation*

Even after the conversion of the behavioural design into a format fully supported for synthesis, there are still other issues to consider, particularly regarding implementation. Figure 6.16 shows an extract from the implementation status report of the Park transform design, targeted at the the Xilinx S40PQ208 FPGA.

```
Design Summary of park transformation structure of figure K:
        Number of errors:       0
        Number of warnings:     5
        Number of CLBs:         244 out of   400   61%
        CLB Flip Flops:     188
        4 input LUTs:       383
        3 input LUTs:       32 (12 used as route-throughs)
        Number of bonded IOBs:      82 out of   160   51%
        IOB Flops:          0
                                    IOB Latches:        0
        Number of clock IOB pads:   1 out of    8   12%
        Number of primary CLKs:     1 out of    4   25%
        Number of secondary CLKs:   3 out of    4   75%
        Number of RPM macros:       17
       Total equivalent gate count for design: 5147
       Additional JTAG gate count for IOBs:    3936
 Writing design file "map.ncd"...
```

Figure 6.16 - *Synthesis report for the hierarchy entity: Park transform*

6.3 Current Model

The current model represents the core module of the field oriented controlled a.c. induction motor drive. This module takes as inputs i_{sd} and i_{sq} plus the speed of the motor. The current model requires the implementation of the following motor equations in the d,q reference frame.

$$i_{sd} = T_r \frac{di_{mr}}{dt} + i_{mr} \qquad \textbf{6-7}$$

$$\text{therefore} \quad \frac{di_{mr}}{dt} = \frac{1}{T_r}(i_{sd} - i_{mr}) \qquad \textbf{6-8}$$

Equation **6-8** can be discretized as follows:

$$i_{mr_{(k+1)}} = i_{mr_k} + K_r(i_{sd_k} - i_{mr_k}) \quad \text{where} \quad K_r = \frac{T}{Tr} \qquad \textbf{6-9}$$

A VHDL code is written to implement the magnetising current model. The block diagram of Figure 6. 17 consists of subtracter, register, multiplication and adder components. The entity named mag_curr has three input ports (*ids, clk, reset*) and one output port *imr*.

```
Entity Mag_Curr is
Port ( clk,reset: in std_logic;
        ids:in std_logic_vector (11 downto 0);
        imr:out std_logic_vector (11 downto 0));
End Mag_Curr;
```

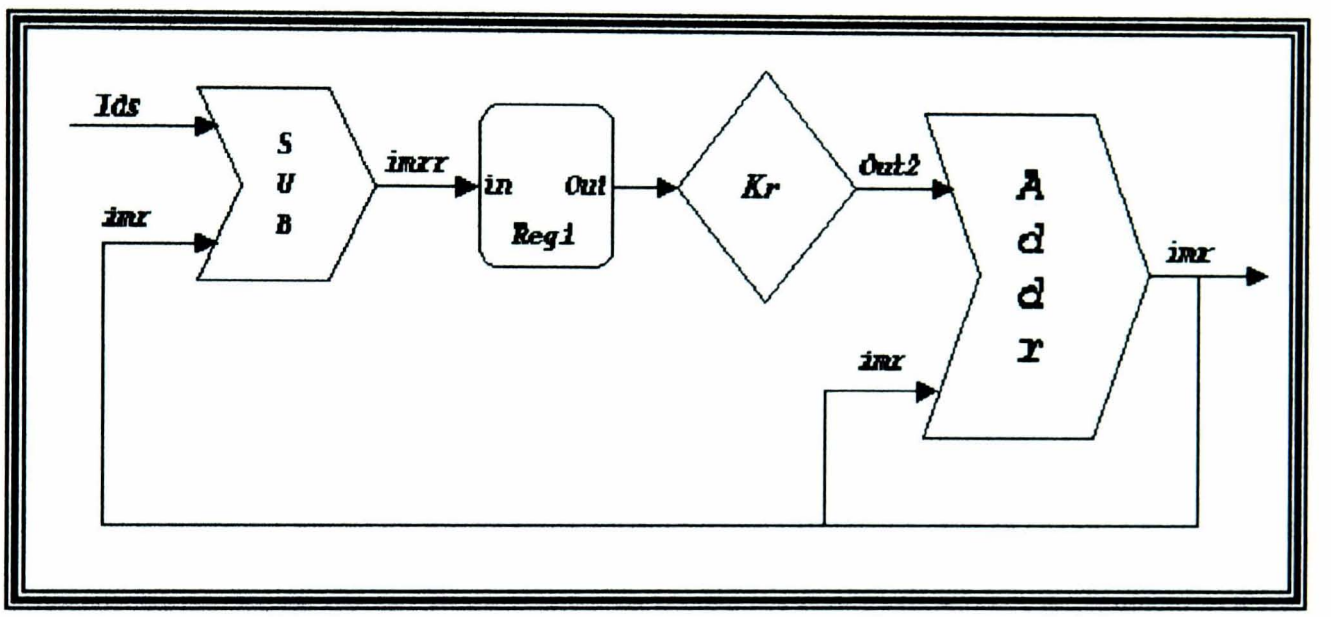


Figure 6. 17- *Block diagram of the magnetising current model*

Ports *clk* and reset are of type std_logic and as such they can only take on one of two values, either a '0' or '1'. Ports *ids* and *imr* are of type *std_logic_vector*. *Std_logic_vector* is a one dimensional array of bits. The width of the array is determined in the port declaration. In the *Mag_curr* entity the width of *ids* and *imr* is defined as 12 bits.

```
Architecture Mag_Curr OF Mag_Curr IS
        {signal declaration }
begin
 process (clk,reset)
       { Variable Declaration which are used as an internal parameter}
    begin
    if reset='1' then
        { Initializing all variables and signals to zeros}
    elsif (clk='1' and clk'event) then
        { Main VHDL code to calculate the magnetizing current}
 end process;
end Mag_Curr;
```

The algorithm shows that the process has a sensitivity list of two signals *clk* and reset. If an event occurs on either of these two signals the process will be activated and the statements within it will be executed sequentially. Since both signals in the sensitivity list are of type *std_logic*, an event would occur if either one of those signals went from a '*Low*' to a '*High*' or vice versa. Once the process has been activated, the sequential if-then-else statement is evaluated and executed. The if statement checks to see that the reset signal is '1'. If so, all internal variables are set to zero. If the reset signal is not equal to '1' the else clause elsif is evaluated and checks to see if *clk* went from a '0' to '1'. If so, the main VHDL code to calculate the magnetising current *imr* is executed.

The rotor flux speed can be expressed as:

$$f_s = \frac{1}{\omega_b}\frac{d\vartheta}{dt} = \omega_r + \frac{i_{sq}}{T_r i_{mr} \omega_b} \qquad \textbf{6-10}$$

If the constant $\frac{1}{T_r \omega_b}$ of the equation above is renamed K_t, equation **6-10** can be rewritten as:

$$f_s = \omega_r + k_t \frac{i_{sq}}{i_{mr}} \qquad \textbf{6-11}$$

where ω_b is the electrical nominal rotor flux speed, ϑ is the rotor flux position, i_{mr} is the magnetizing current, and $T_r = \frac{L_r}{R_r}$ is the rotor time constant. Knowledge of this constant is critical to the correct functioning of the current model. Figure 6.18 illustrates the steps required to calculate the rotor flux speed, which will be integrated to get the rotor flux position.

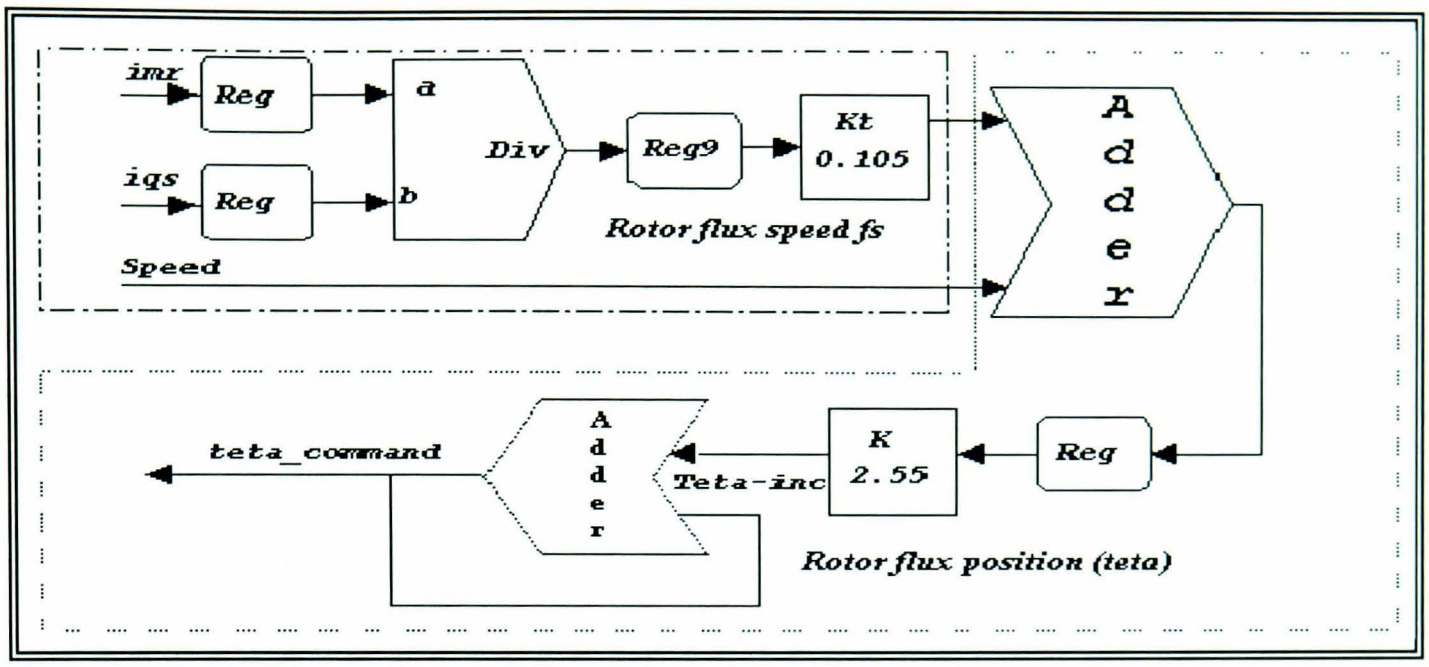


Figure 6.18 - *Block diagram of digital presentation of rotor flux speed and rotor flux position*

Once the rotor flux speed f_s has been calculated, the necessary rotor flux position theta_command (ϑ) is computed by the integration formula:

$$\vartheta_{K+1} = \vartheta_K + \omega_b f_s T \qquad \textbf{6-12}$$

As the rotor flux position range is (0 to 2π), 12 bit *std_logic_vector* values have been used to achieve good resolution. Figure 6.19 demonstrates the relationship between the flux position and its numerical representation:

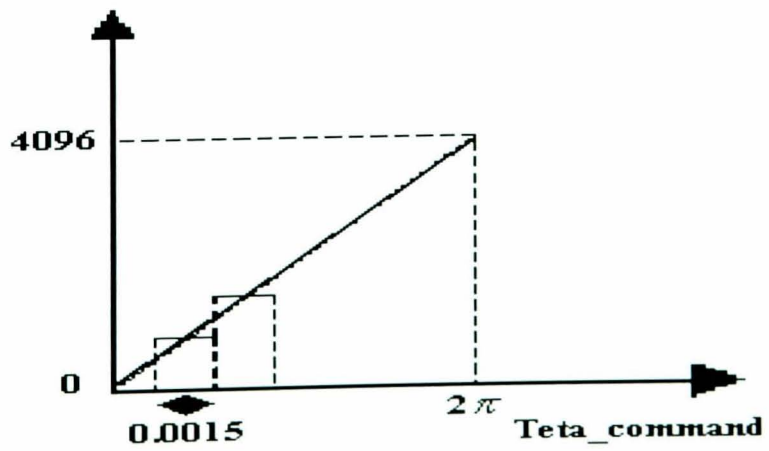


Figure 6.19 - *Link between rotor flux position and its numerical representation*

In equation **6-12** let $\omega_b f_s T$ be called ϑ_{inc}. This variable is the angle variation within one sample period. At nominal operation (in other words when f_s=1, and the mechanical speed is equal to the nominal speed of the motor) ϑ_{inc} is thus equal to:

$$\vartheta_{inc} = \omega_b f_s T \qquad \textbf{6-13}$$

where T is the sampling time. If T = 100 µs then

$$\vartheta_{inc} = 2*\pi*f_n*f_s*T \quad = 2*\pi*50*1*100*10e^{-6} \quad = \quad 0.031415 \text{ rad.}$$

In one mechanical revolution performed at nominal speed there are $\frac{2\pi}{0.031415} \approx 200$ increments of the rotor flux position. Let K be defined as the constant that converts the (0 to 2π) range into the (0 , 4096) range. K is calculated as follows:

$$\text{K} = \frac{4096}{200} = 20.48$$

With the help of this constant, the rotor flux position computation and its formatting becomes

$$\vartheta_{K+1} = \vartheta_K + Kf_s \qquad \textbf{6-14}$$

The theta_command (ϑ) variable is thus represented in VHDL as a 12 bit *std_logic_vector*. This position is used in the transformation modules as the entry point in the sin look_up table.

A VHDL code is written to implement the block diagram illustrated in Figure 6.18, which represents the digital circuit calculating the rotor flux speed and the rotor flux position defined by equations **6-11** and **6-14**. To implement equation **6-11**, which calculates the rotor flux speed f_s using VHDL a division operation is required. This is behaviorally achieved with the following VHDL statements:

```
PROCESS(next_step)
variable wrr,Tr, isq,wmr :real:=0.0;
  imr,imr1,isd    :real:=0.0;
begin
  imr1:=(isd-imr)/Tr;
          imr:=imr+imr1*dt;
          if imr >0.0 then
      wmr:= wrr+(isq /(Tr*imr));
End process;
```

The division operator '/' is not supported by present synthesis tools in Xilinx Foundation 1.5 [69]. Therefore, in order to implement this calculation in hardware, it is necessary to design a digital divider at structural level.

6.4 The Divider

Equation **6-11** shows that the current model process requires a division operation. The division is a repeated process of compare, right shift and subtract operation. Subtraction is usually carried out using 2's complement representation. The assumption is made that the dividend X is n bits long, the divisor Y is m bits long, X_m denotes the m most significant bits of X, $m \leq n$. The division process starts by comparing Y with X_m.

If $Y > X_m$, then, compare Y with X_{m+1} until $X_{m+i} \geq Y$. At this point, '1' should be entered as the most significant bit (msb) of the quotient Q. Y is then right shifted i places and subtracted from X_{m+i}. The m+i+1 msb of X is appended to the partial reminder. If the partial remainder is greater than Y, the next msb of Q is 1, and Y is shifted right one place and subtracted from the partial remainder. Otherwise, the process outlined is repeated, with 0's placed into the appropriate bits of Q. This process continues until all n bits of X have been exhausted, at which point the last remainder will be obtained. The sign of the quotient is determined from the signs of the dividend and divisor. If they are the same, the quotient is positive; otherwise, the quotient is negative [70].

To illustrate the division process, consider the following example. The signs of the numbers in this example are assumed to be the same and have been omitted.

	Dividend	**Divisor**	**Quotient**
	X	Y	Q
◆ Compare five (msb) of X with Y	0110110110 /	10001 =	11001
X < Y.	01101		
◆Compare Six bits of X : X > Y;	011011		
	- 10001		
shift right Y and subtract;			
enter msb of Q = 1.	01010		
◆ Partial remainder > Y;	010100		

shift right Y and subtract;	- 10001
enter 1 in Q.	00011
◆ Partial remainder < Y;	000111
shift right; enter 0 in Q.	
◆ Partial remainder < Y;	001111
shift right; enter 0 in Q	
◆ Partial remainder > Y;	011110
shift right Y and subtract;	- 10001
enter 1 in Q.	01101 (last remainder).

Note that if the partial remainder is greater than Y then enter 1 in the quotient otherwise enter 0.

In the behavioural design, variables isq, *Tr* and imr are declared as type *REAL*, which has no meaning in hardware design. All variables and signals should be given a well defined range such as an "integer range 0 to 512" or a "*std_logic_vector*" type. In order to design the divider, the entity named dividera has been created, which consists of four input ports (*clk*, *reset*, *A*, B and one output port (*Div*):

```
Entity dividera is

   Generic( n: in integer:=13
            m: in integer:=12);
   Port (  CLK: in std_logic;
         Reset: in std_logic;
         A: in std_logic_vector(n-2 downto 0);
          B: in std_logic_vector(m-1 downto 0);
         ready: out std_logic;
         Div: out std_logic_vector(m-1 downto 0));

End dividera;
```

The code is written to implement the above algorithm for a 12 bit dividend (*A*) and a 12 bit divisor (*B*). The entire operation is carried out in a VHDL process. In order to avoid division by zero the following statement is included inside the Process:

```
Process (Clk,Reset)
begin
  if B = '0' then
        -- Avoid division by zero: assign Div=0
          Div<=(others=>'0');
  End if;
End process;
```

The Process does not become active until one of the signals (clk or Reset) in its sensitivity list changes value. A Generic statement is used within the entity declaration, which has a constant value and can therefore be substituted into the associated design unit. The use of a Generic in this format makes the entity Dividera to be adopted by any number of bits.

Figure **6.20** illustrates the input stimuli applied to the divider and the corresponding outputs.

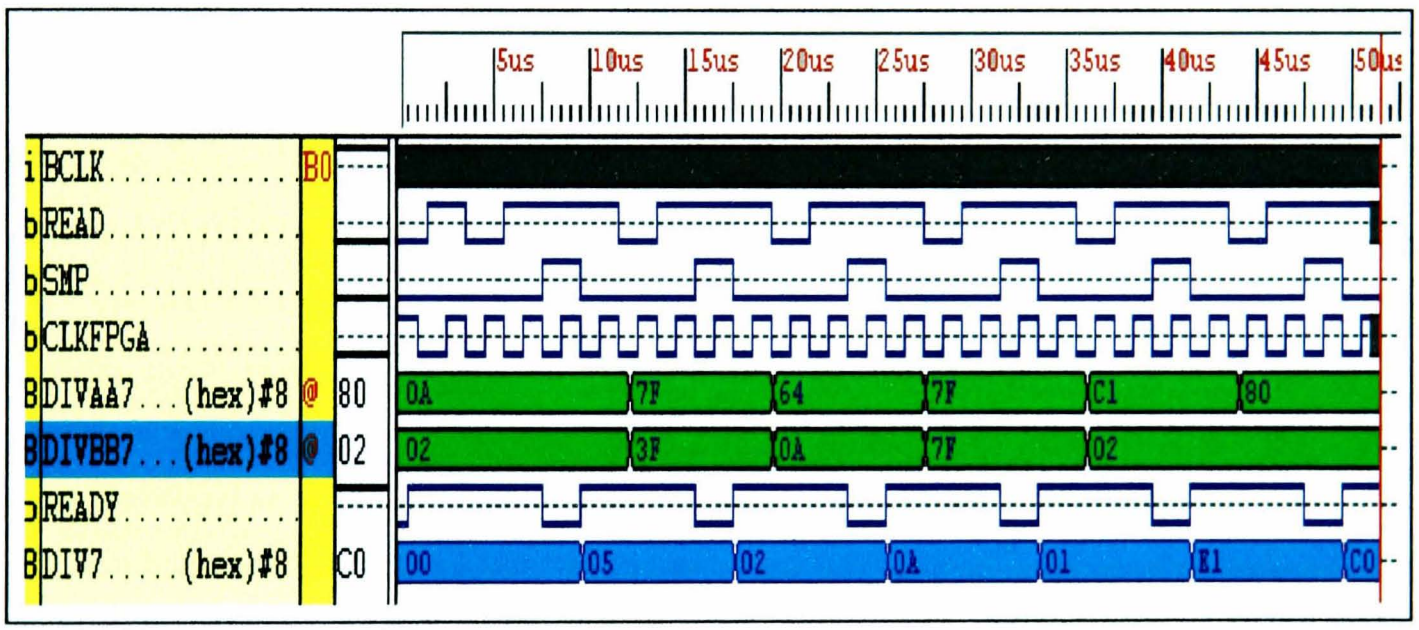


Figure 6.20 - ***Simulated waveform of the divider***

6.5 Generation of Sine and Cosine Values

The sinewave generation process requires a look-up table Appendix F to store information about the shape of the output waveforms. The look_up table contains the samples of the sinwaves to be generated and the table is read in sequence. As a compromise between the position accuracy and the used memory minimisation, this table contains 2^8=256 words to represent the (0 , 2π) range. In order to have the cosine value, 256/4 = 40h must be added to the sine index. Part of the VHDL code to address the sine look up table is given below (the complete code is included in Appendix B).

```
Entity sin_rom IS
 Port( clk,reset,enable : in std_logic;
   A : in  std_logic_vector(9 downto 0);
   Sin : out  std_logic_vector(7 downto 0);
   Cos : out std_logic_vector(7 downto 0));
End sin_rom;
... ... ... ...
Begin
 Process  (A,B,...)
 Begin
    B<=A+co;   --    Where co = 512 / 4 = 80h
    Sin  <= V1 (conv_integer(A));
    Cos <= V1 (conv_integer(b));
 End Process;
... ... ... ... ... ... .
End;
```

The Entity *sin_rom* is a look up table Appendix F. It has one 9 bits input bus *A* and two 8 bits output buses, sine and cosine. The input bus comes from the current model entity and it is used as an index to access the look up table. The VHDL model of the look up table contains only one process that associates each address value with a set of 8 bits in vector V1. The *std_logic_vector* input *A* is transformed into an integer using *conv_integer* function. This integer is used as an index for the vector V1. Figure 6.21 shows the simulated waveform for the *sin_rom*. It can be seen that the code is working correctly, as the output (sine and cosine) do change as soon as the input index (*A*) changes.

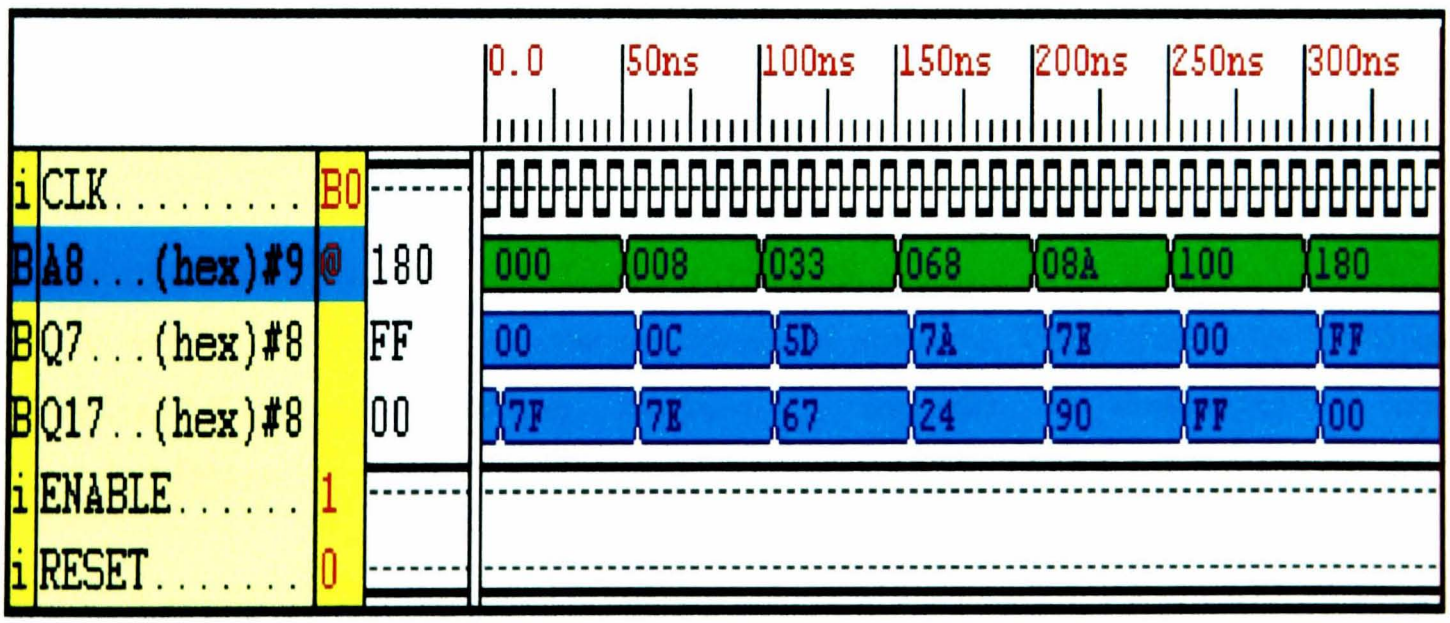


Figure 6.21 – Simulated waveforms of the sin rom

6.6 Overall Simulation of the Current Model

Figure 6.22 shows the waveforms of all the signals in the current model. It can be noticed that there is an event on signal *START* at approximately 20 µs changing its value from '1' to '0'. This causes i_{ds}, i_{qs} and the magnetising current i_{mr} to be executed and new values to be scheduled on these signals. Once a new value is obtained for the magnetising current, an event on signal *STARTC* from '1' to '0', occurs. This causes *Div*, *QA* and *QB* to be executed and new values to be also scheduled on these signals. *QA* and *QB* represent *sin_theta* and *cos_theta* respectively.

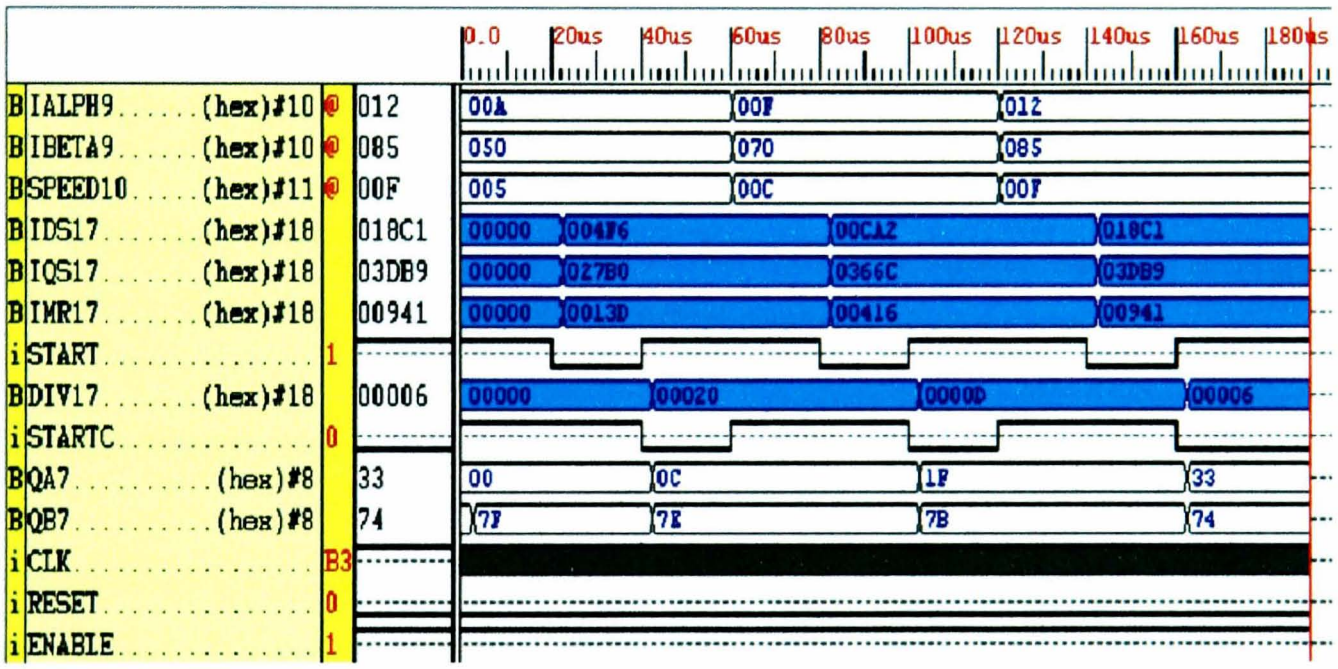


Figure 6.22 - *Simulated waveforms of the current model*

Some values were taken as examples to verify the simulation results shown in Figure 6.22 for the digital circuit presenting the current model structure. These results are based on equations **6-5**, **6-6** and **6-9**. When *ialph* = A , *ibeta*=50 *sin_theta* (QA) = 0 and *cos_theta* (QB) = 7F then

$i_{ds} = A * 7F + 50 * 0 = \underline{\mathbf{4F6}}$

$i_{qs} = -A * 0 + 50 * 7F = \underline{\mathbf{27B0}}$

$i_{mr} = i_{mr} + 0.25\ (i_{ds} - i_{mr})$

i_{mr} = 0 + 0.25 (4F6 - 0) = **13D**

i_{ds} = F * 7E + 70 * C = **CA2**

i_{qs} = -F * C + 70 * 7E = **366C**

$i_{mr} = i_{mr} + 0.25\ (i_{ds} - i_{mr})$

i_{mr} = 13D + 0.25 (CA2 - 13D) = **416**

i_{ds} = 12 * 7B + 85 * 1F = **18C1**

i_{qs} = -12 * IF + 85 * 7B = **3DB9**

$i_{mr} = i_{mr} + 0.25\ (i_{ds} - i_{mr})$

i_{mr} = 416 + 0.25 (18C1 – 416) = **940**

The underlined results can also be seen in the waveform viewer window in Figure 6.22. This shows that the digital circuit of the current model is working correctly.

6.7 PI Controllers

PI controllers are universally known because of their flexibility combined with relatively easy tuning. This brief introduction describes the conversion from the continuous to the discrete time domain, essential for implementation on a digital processor. The vector control algorithm includes four Proportional Integral (*PI*) controllers to control speed, flux, i_{qs} (torque producing current), and i_{ds} (flux producing current).

The control law is given by the equation

$$u = K_P .e + K_I . \frac{1}{T} \int_0^t e.dt \qquad \textbf{6-15}$$

Where

e is the error signal.

u is the control signal and

Differentiating equation **6-15** gives

$$\frac{du}{dt} = K_P \cdot \frac{de}{dt} + K_I \cdot e \qquad \textbf{6-16}$$

The discrete time difference equations which implement the PI controller are:

$$I(k) = I(k-1) + k_i \cdot e(K) \qquad \textbf{6-17}$$

$$U(k) = kp \cdot e(K) + I(k) \qquad \textbf{6-18}$$

Where

k_I	=	integral constant
I (k)	=	integral of error.
I(k-1)	=	previous integral of error.
e (K)	=	error.
Kp	=	proportional constant
U(k)	=	controller output.

The discrete time PI controller is shown in block diagram in Figure 6.23.

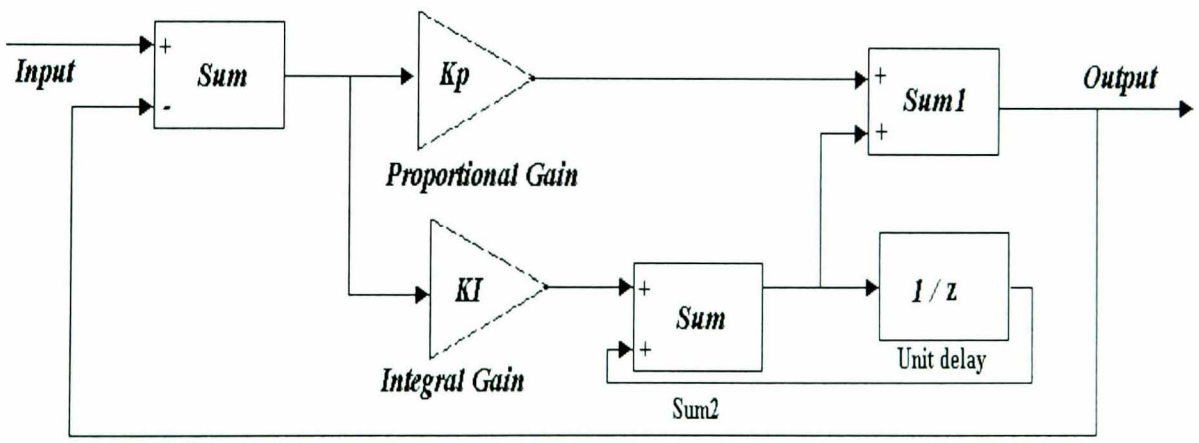


Figure 6.23 - *Discrete time PI controller*

The transition from the continuous to the discrete time domain entails that the integral operation is approximated by a discrete summation. There are several methods for replacing the integral. Two of them are discussed here.

6.7.1 Zero Order Hold (ZOH)

In this approach, the signal is sampled at the instant K and held constant until the next sampling instant K+1. Figure 6.24 illustrates this.

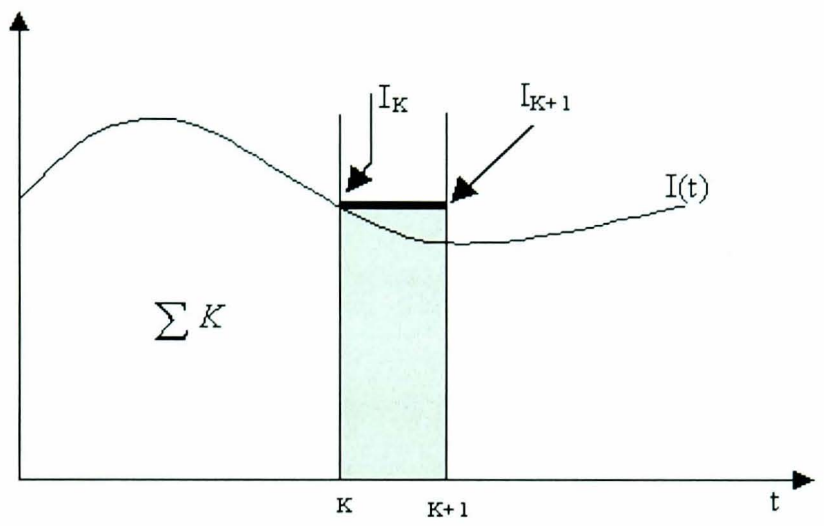


Figure 6.24 - *Integral with ZOH approximation*

This integral operation is approximated by accumulating the rectangular areas. Denoting the sum at instant K with $\sum K$, the signal at instant K with I_K and the sample time with T_{sample}, the " integration" is achieved by:

$$\sum_{K+1} = \sum_{K} + I_K . T_{sample} \qquad \textbf{6-19}$$

the same equation may be expressed in the z-domain by:

$$z.\sum(z) = \sum(z) + I(z).T_{sample} \qquad \textbf{6-20}$$

6.7.2 First Order Hold (FOH)

A slightly improved approach is shown in Figure 6.25.

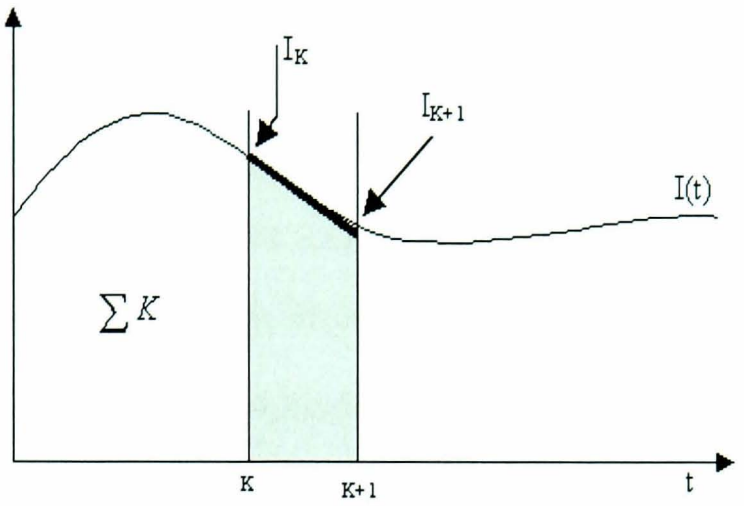


Figure 6.25 - *Integral with FOH approximation*

The integral operation is approximated by accumulating the trapezoidal areas. Denoting the sum at instant K with $\sum k$, the signal at instant K with I_K and the sample time with T_{sample}, the "integration" is achieved by:

$$\sum_{K+1} = \sum_{K} + \frac{I_K + I_{K+1}}{2}.T_{sample} \qquad \textbf{6-21}$$

Where the last term is derived from the known formula for the area of a trapezoid. Following a similar procedure as in the previous section, this may be expressed in the z-domain by:

$$z\sum(z) = \sum(z) + I(z).\frac{1+z}{2}.T_{sample} \qquad \textbf{6-22}$$

In discrete time system, Equation **6-16** can be written as:

$$U(kT) - u(kT-T) = K_p .\{e(kT) - e(kT - T)\} + K_I .e(kT)$$

$$\Delta u = K_p .\Delta e + K_I .e \qquad \textbf{6-23}$$

where

Δu is the change in u over one sampling period and

Δe is the change in e over one sampling period.

6.7.3 PI Regulators

The i_{sd} and i_{sq} components of equations **6-5** and **6-6** are compared to the references i_{sdref} (the flux reference) and i_{sqref} (the torque component). The torque command i_{sqref} corresponds to the output of the speed regulator. The flux command i_{sdref} is the output of the flux controller illustrated in Figure 6.29. The current regulator outputs are v_{sdref} and v_{sqref} and they are applied to the inverse Park transformation. The outputs of this projection are $v_{s\alpha ref}$ and $v_{s\beta ref}$, the components of the stator voltage vector in the α, β orthogonal reference frame. These are the inputs of the space vector PWM. The outputs of this block are the signals that drive the inverter.

According to [71], the limiting point is that during normal operation or during the tests, large reference value variations or large disturbances may occur, resulting in saturation and overflow of the regulator variables and output. If they are not controlled, this kind of non linearity damages the dynamic performance of the system. To solve this problem, a solution is to put a correct the integral component as shown in Figure 6.26. The simulation result is given in Figure 6.28.

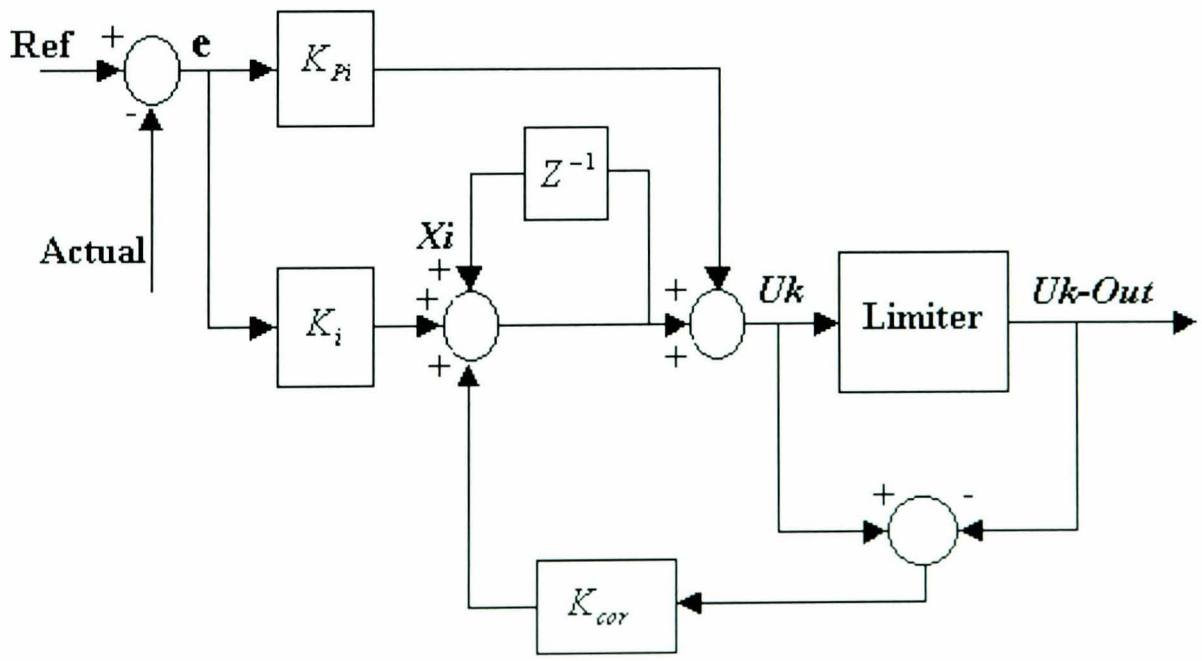


Figure 6.26 - *Numerical PI regulator with correction of the integral term*

In order to verify the numerical PI regulator with correction a small program is written to simulate the digital control system with the continuous process and the discrete PI

controller as illustrated by Figure 6.27. The PI term correction algorithm in a high level language is given below:

$$e = Ref - Actual$$

$$U_k = X_i + K_{pi} * e$$

$$U_k\text{-}out = U_k$$

$$If\ U_k > U_{max}\ THEN\ U_k\text{-}out = U_{max}$$

$$If\ U_k < U_{min}\ THEN\ U_k\text{-}out = U_{min}$$

$$E1 = U_k - U_k\text{-}out$$

$$X_i = X_i + K_i e + K_{cor} e$$

with U_{max} and U_{min} refer to the limitations of the output variable.

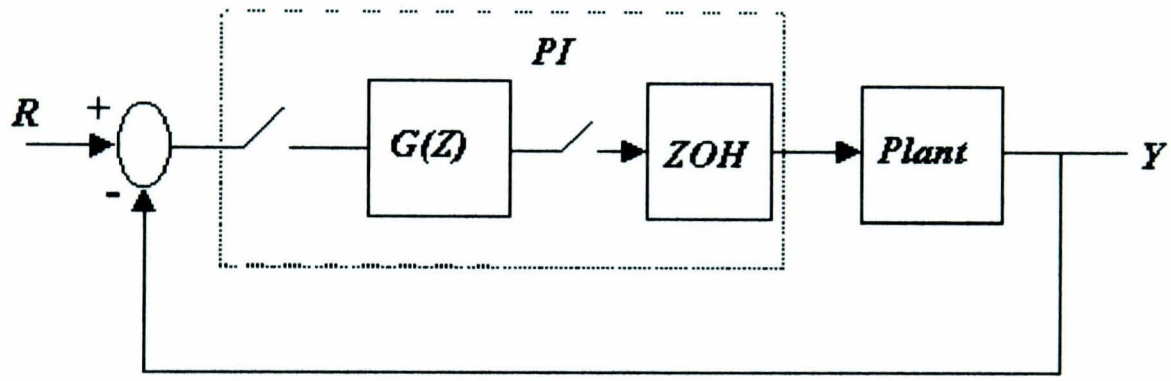


Figure 6.27 - *The block diagram of the digital control system*

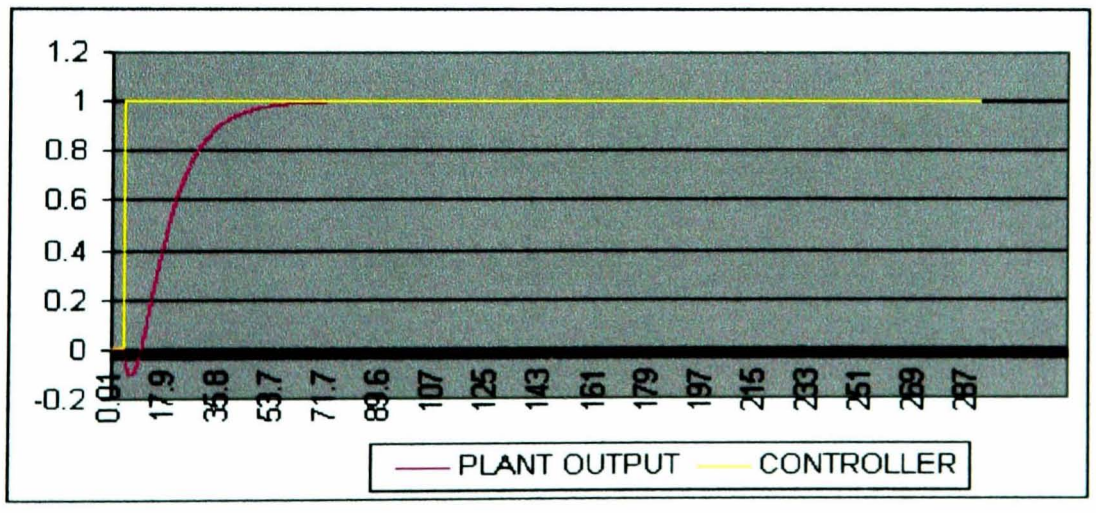


Figure 6.28 - *Simulated results with a correction of the integral component*

Equation **6-23** consists of a sequence of multiply and accumulate operations that are ideally suited for implementation in FPGA or DSP .

The PI controller is designed and modelled in an EDA environment using VHDL. The model is broken down into four controllers, each performing a specific function. These are: speed controller, flux controller, current producing flux controller and current producing torque controller, as illustrated in Figure 6.29. The result is that the design of each controller can be modified or simply combined with another one to form a complete system.

The VHDL design of the controller is configured with the entity name PI controller. It is designed as a synchronous circuit with nine input signals including Clk and Reset. Four output signals provide simulated information on the current and voltage references. Figure 6.29 shows a diagram of the controller structure.

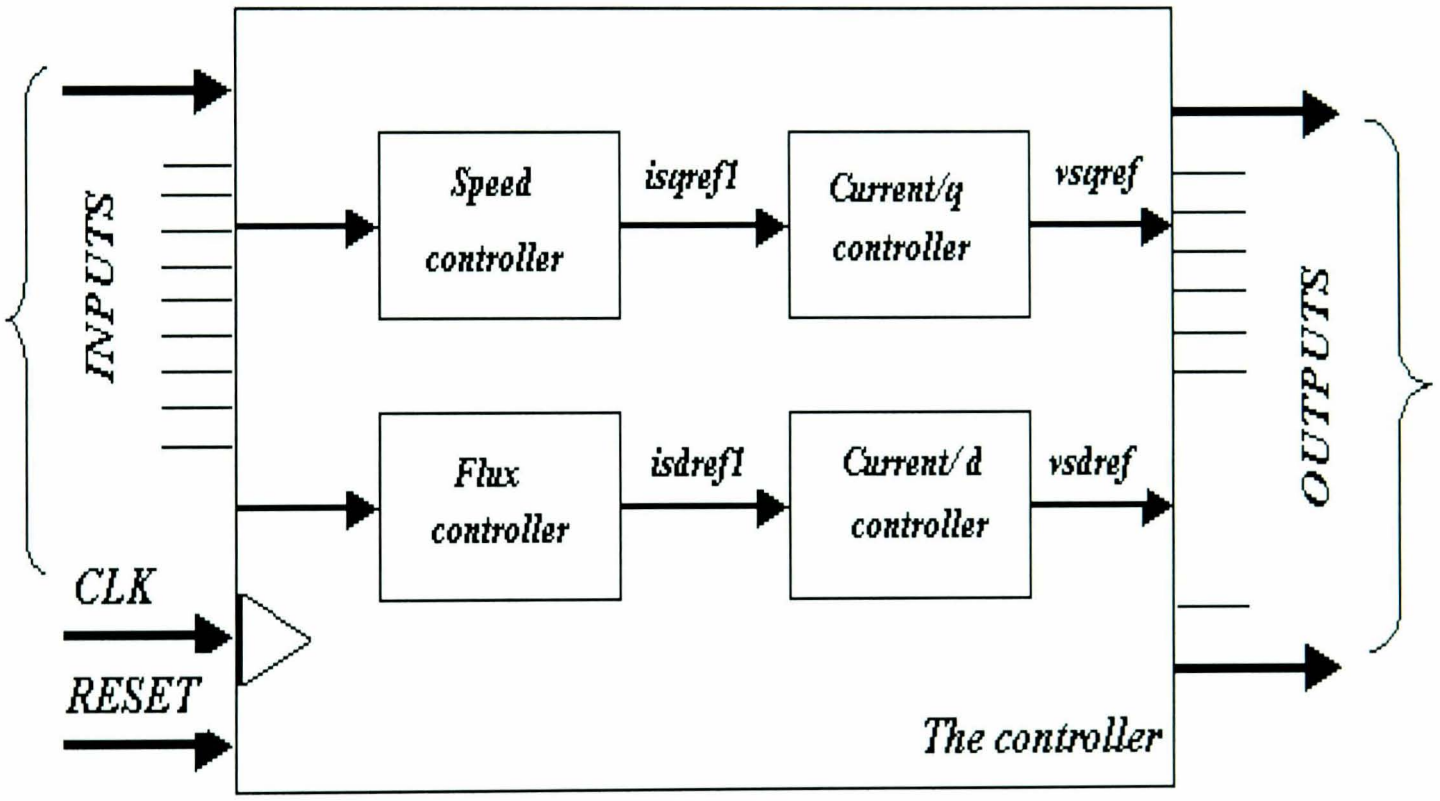


Figure 6.29 - *The controller structure*

A VHDL code is written to implement the controller structure . The complete listing of the code is included in Appendix (B). The entity named PI controller has nine input ports (*clk, reset, readys, speed, refspeed, isq, isd, imref, imr*) and four output ports (*isdref, isqref, vdref, vqref*).

```
Entity PI_regulator IS

port ( clk,reset,readys: in std_logic;

        speed:in std_logic_vector (11 downto 0);
        isq:in std_logic_vector (11 downto 0);
        isd:in std_logic_vector (11 downto 0);
        im:in std_logic_vector (11 downto 0);
        imref:in std_logic_vector (11 downto 0);
        refspeed:in std_logic_vector (11 downto 0);
        isdref:out std_logic_vector (11 downto 0);
        vsdref:out std_logic_vector (11 downto 0);
        isqref:out std_logic_vector (11 downto 0);
        vsqref:out std_logic_vector (11 downto 0));

End PI_regulator;

Architecture behavioral OF PI_regulator IS

{ ..... Constant declaration ..... ..... .....}

Begin

    Process (clk,reset,speed,isq,isd,im,imref,refspeed)

        {..... Variable Declaration ..... };

        {..... ..... Speed Controller ..... .....};

        {..... ..... ..... Flux Controller ..... ..... .....};

        {..... ..... ..... Current/d Controller ..... ..... .....};

        {..... ..... ..... Current/q Controller ..... ..... .....};

End Process;

End behavioral;
```

A series of computer simulations were conducted on the *PI_controller* to analyse its performance before proceeding to hardware implementation. Simulated waveforms of the *PI_controller* operation are shown in Figure 6.30.

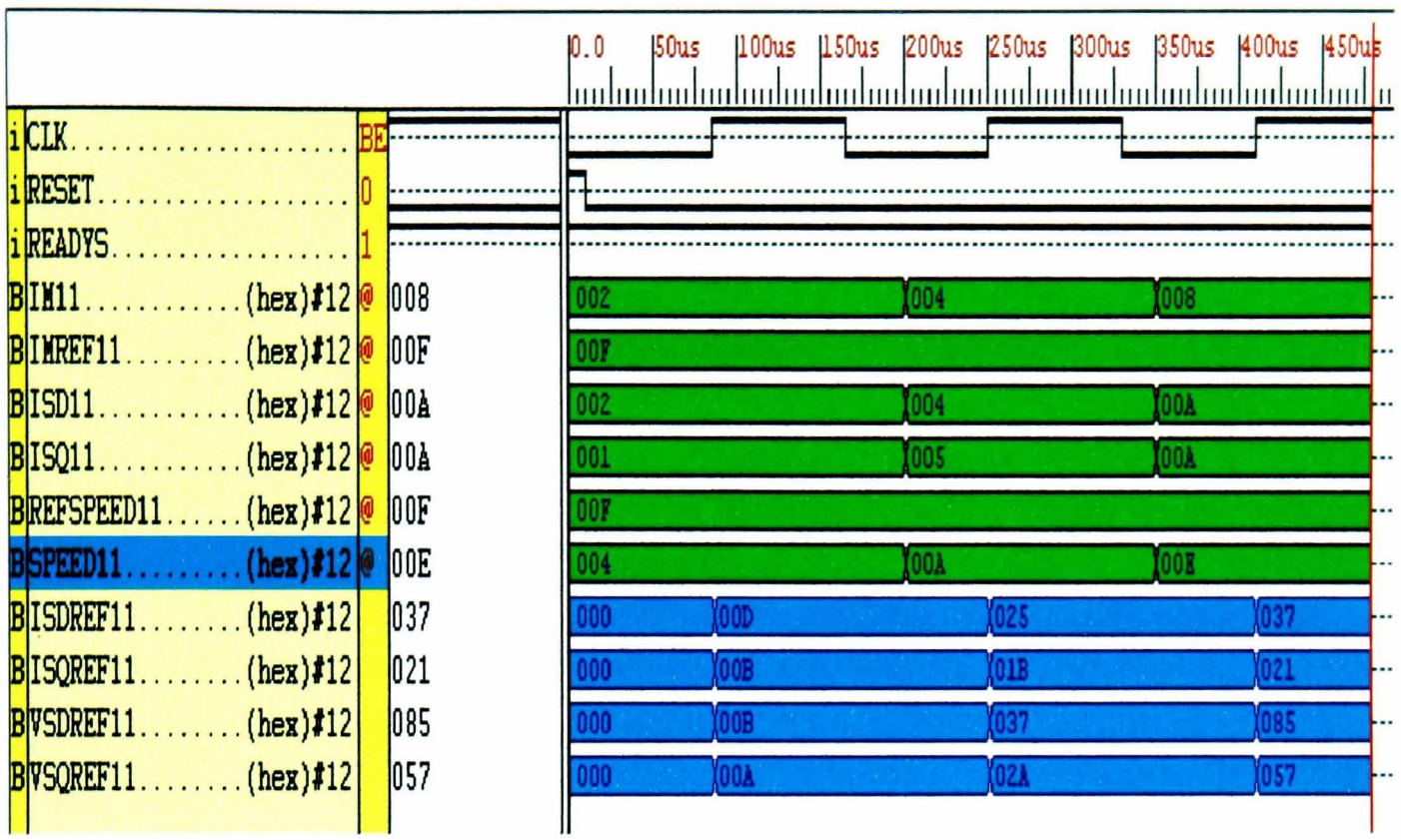


Figure 6.30 - *Simulated waveform for PI controller*

6.8 PWM Waveform Generator

Due to the rapid progress in motor control and microelectronics technologies, the development of universal a.c. drives has become a major trend. Although most a.c. drives in use today adopt microprocessor based digital control strategy, implementation of current control loop and PWM control are still tied to analogue control circuitry as shown in Figure 6.31.

This kind of control scheme posses the advantage of dynamic response, but suffers the disadvantages of complex circuitry, limited functions, and difficulty in circuit modification. The rapid development in high performance low cost digital signal processor (DSP's) has encouraged research on digital PWM control and digital current control for a.c. drives [72, 73, 74].

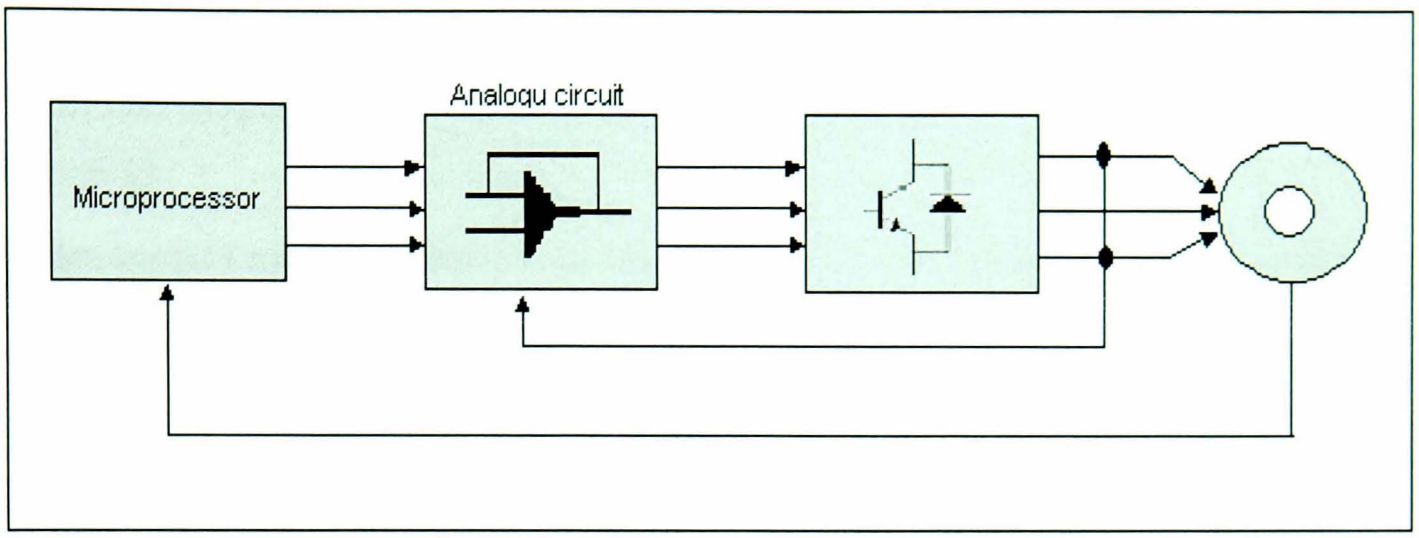


Figure 6. 31- *PWM control structure of digital a.c. drives – analogue*

Figure 6.32 illustrates a typical control architecture of a DSP based a.c. drive. This control scheme has the advantages of simple circuitry, software control, and flexibility in adaptation to various applications. However, generating PWM gating signals and implementing the current control loops require a high sampling rate to achieve a wide bandwidth performance. Therefore, a large amount of DSP computation resources must be devoted to generating the PWM signals and executing the motor current algorithms [75].

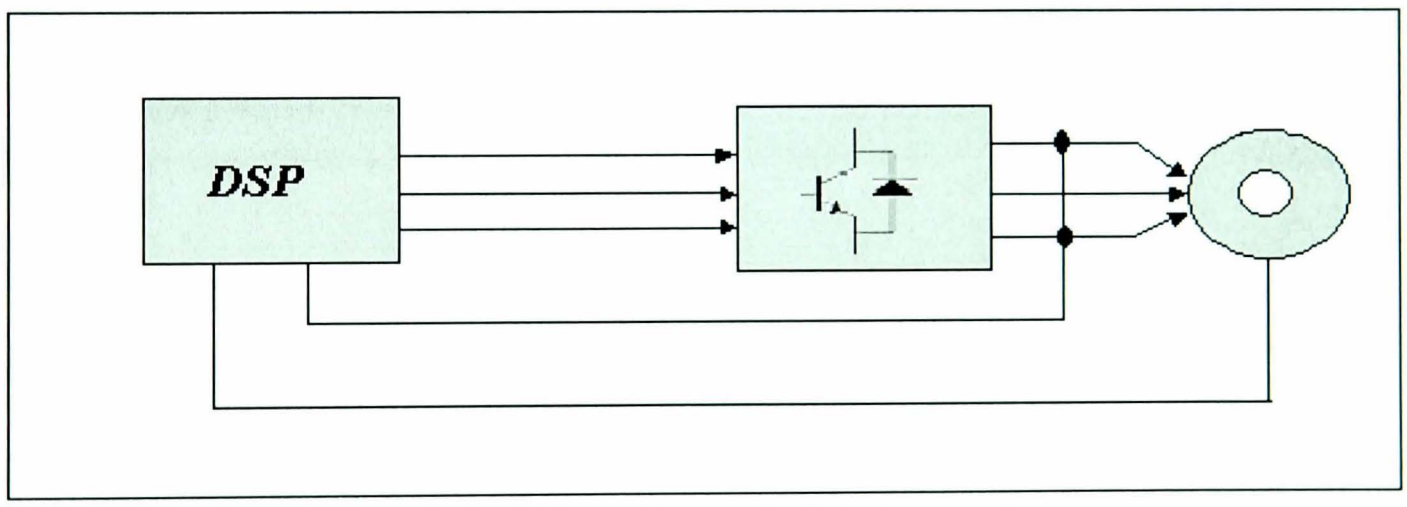


Figure 6.32 - DSP based digital control

Recent developments in the field of microelectronics have enabled complex switching strategies for transistorised inverters to be implemented by means of ASIC technology. Employing FPGA to realize PWM strategies provides advantages such as rapid prototyping, simple hardware and software design, higher switching frequency, and relieving the computation load of microprocessors. The novel digital circuit realization scheme developed for the PWM control IC employing a single FPGA Spartan XCS40 from Xilinx, Inc.

The designed PWM IC may serve either for a.c. motor drives or three phase a.c. voltage regulation systems. It can be incorporated as part of the digital current loop for a.c. motor drives. The structure of the PWM waveform generator circuit adopted is illustrated in Figure 6.33.

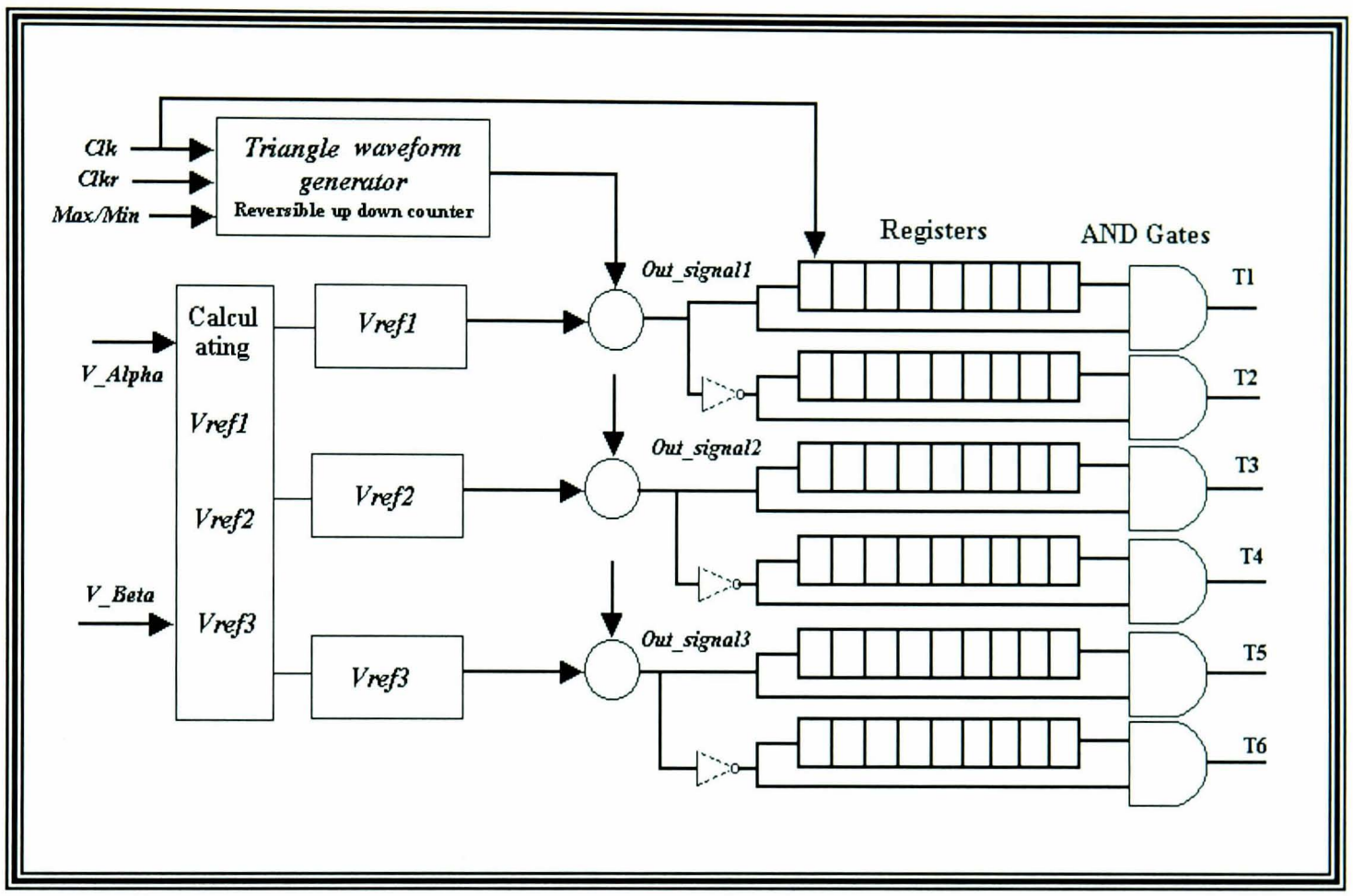


Figure 6.33 - *PWM waveform generator*

The PWM generator operation is controlled by means of *clk* and *clkr* signals alongside the values supplied through the maximum and minimum values external bus. The maximum count values determine the amplitude of the triangular carrier generated by the reversible up-down counter, as illustrated in Figure 6.34 and Figure 6.35. The counter generates successive numbers between min count and max count in increasing and decreasing order, thereby producing the digital equivalent of the triangular carrier signal used by analogue PWM generators. Consequently the period of the digital carrier will be proportional to its amplitude as it can be seen from Figure 6.34 and Figure 6.35.

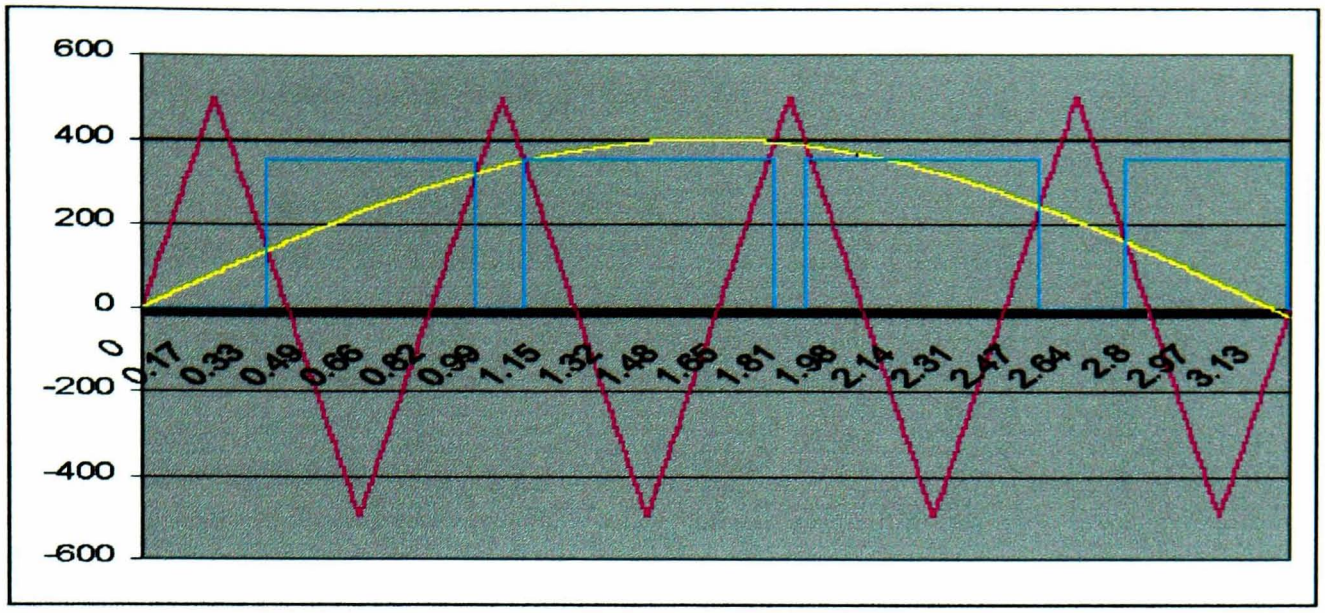


Figure 6.34- *Simulated triangular waveform at a maximum amplitude of 500*

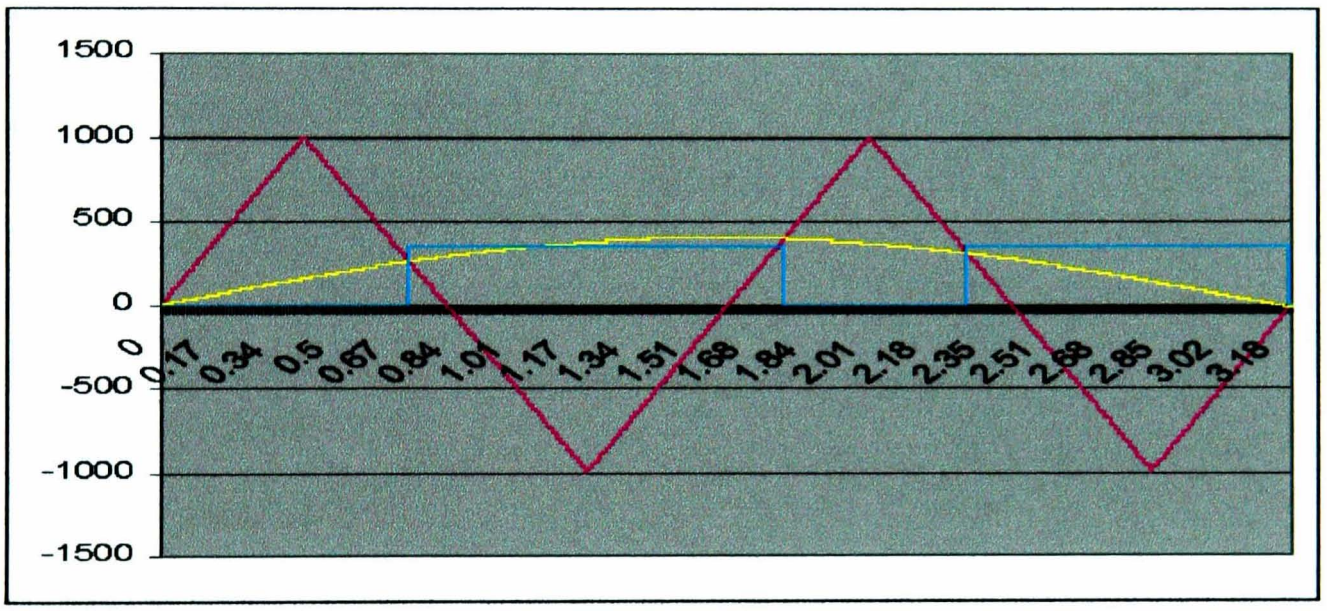


Figure 6.35 - *Simulated triangular waveform at a maximum amplitude of 1000*

The output signal is generated by a digital comparator supplied with the samples of the sinusoidal carrier and the values of the triangular carrier. The sine wave samples are derived from the inverse Park transformation. The VHDL model considers the PWM generator to be a single entity containing two processes. The sequence given is part of the complete VHDL model. It describes the operational logic of the Triangle process unit, which has several tasks to perform inside the circuit. As long as the reset signal is high and direction signal is low, both up-down counters are halted. When a transition occurs and the

reset signal is low, the Triangle process starts working and *out_signal1*, *out_signal2* and *out_signal3* is generated as shown in Figure 6.36.

```
Traingl :Process (clk,reset,vrefa,vrefb,vrefc)
    variable direction :std_logic;
 begin
  if reset='1' then
   Triangl<="00000000000001";
       direction :='1';
  elsif (clk'event and clk='1') then
     if direction ='1' then
        if Triangl<max then
           Triangl<=Triangl+1;
          else
          Triangl<=Triangl-1;
             direction:='0';
           end if;
          else
          if Triangl>min then
             Triangl<=Triangl-1;
             else
             Triangl<=Triangl+1;
            direction:='1';
          end if;
          end if;
        if vrefa< Triangl then      -- vr1 coresponds to either va or vb or vc
           out_signala1<='0';
           else
          out_signala1<='1';
        end if;
       if vrefb< Triangl then       -- vr2 coresponds to either va or vb or vc
           out_signala2<='0';
           else
          out_signala2<='1';
        end if;
        if vrefc< Triangl then      -- vr3 coresponds to either va or vb or vc
           out_signala3<='0';
           else
          out_signala3<='1';
end if;
```

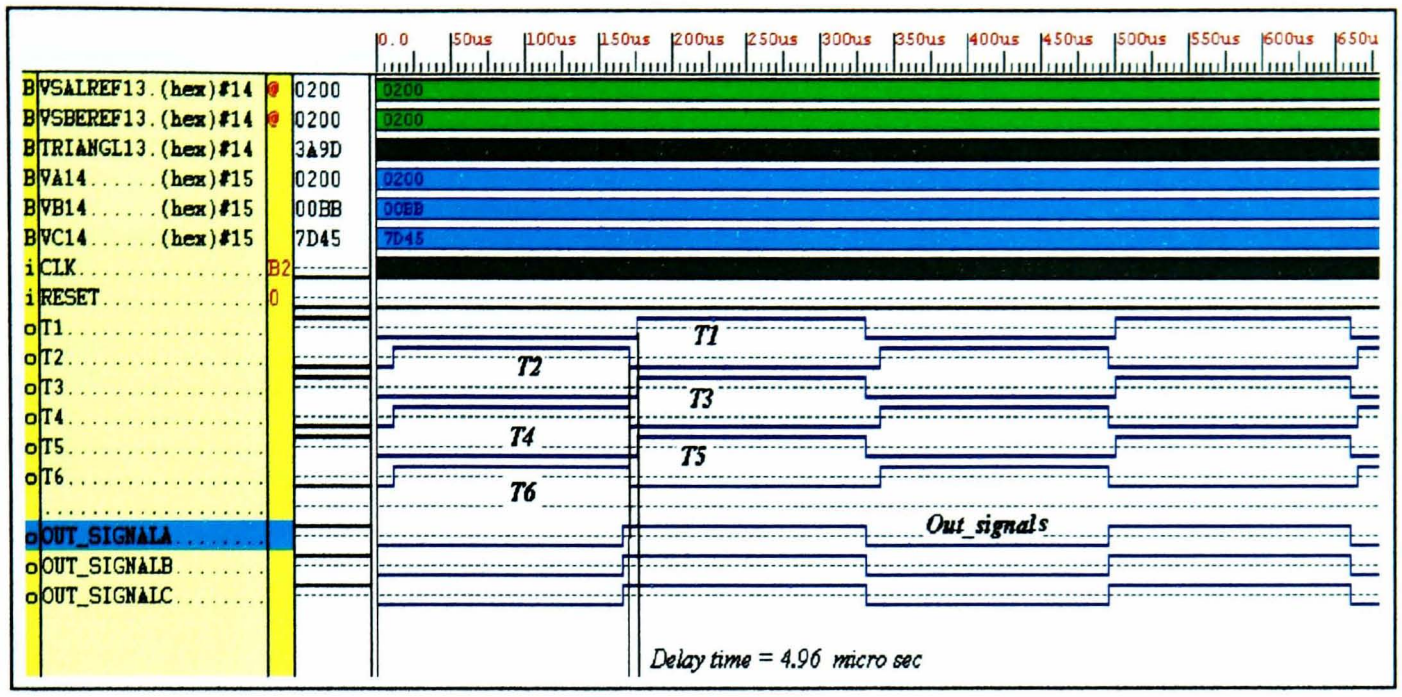


Figure 6.36- *Simulated waveform for PWM*

The second process is to generate the correct signal to control the transistors in the PWM inverter. The out_signals defining the desired output voltage of the PWM inverter are transformed into six logic signals. Each amplified and applied to one power transistor. The edges of these signals are shifted by 4.9 µs so that short circuits are avoided between transistors on the same inverter phase (*T1* and *T2*) as illustrated in Figure 6.36. Figure 6.37 to Figure 6.39 show a simulated waveform for only two transistors, clearly indicating the delay time between the two signals.

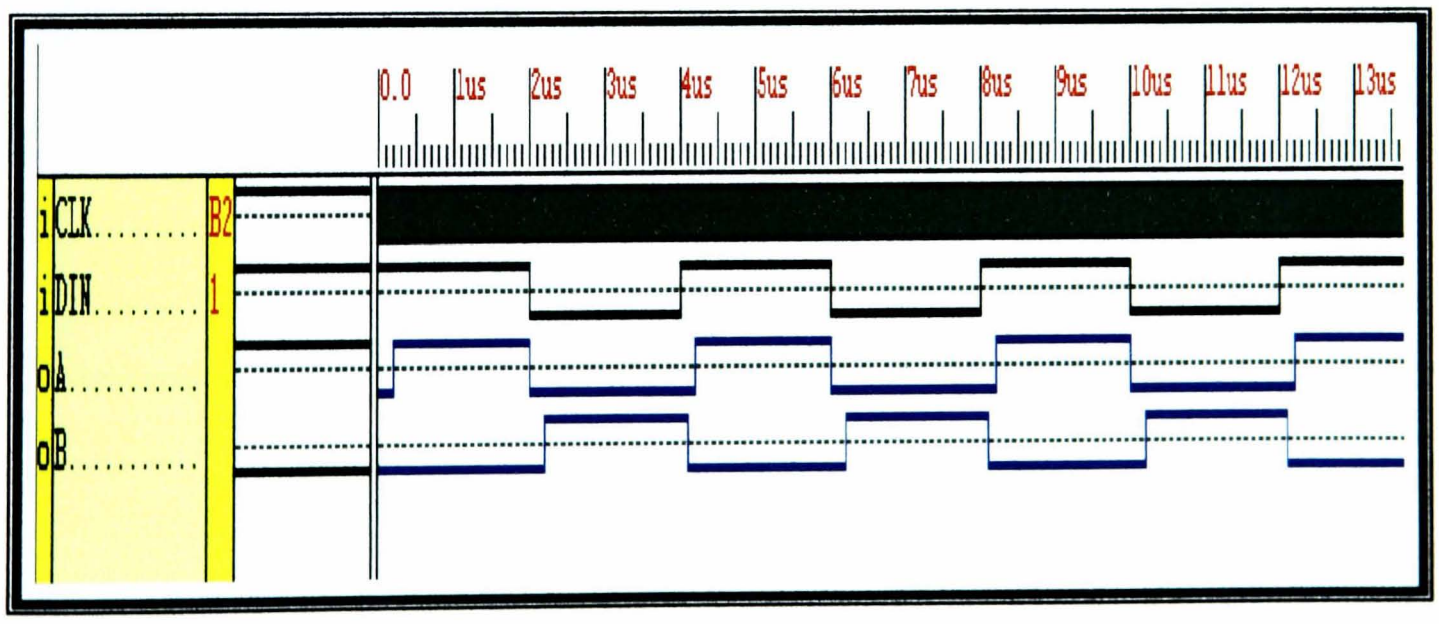


Figure 6.37 - *Simulated waveform for two transistors PWM*

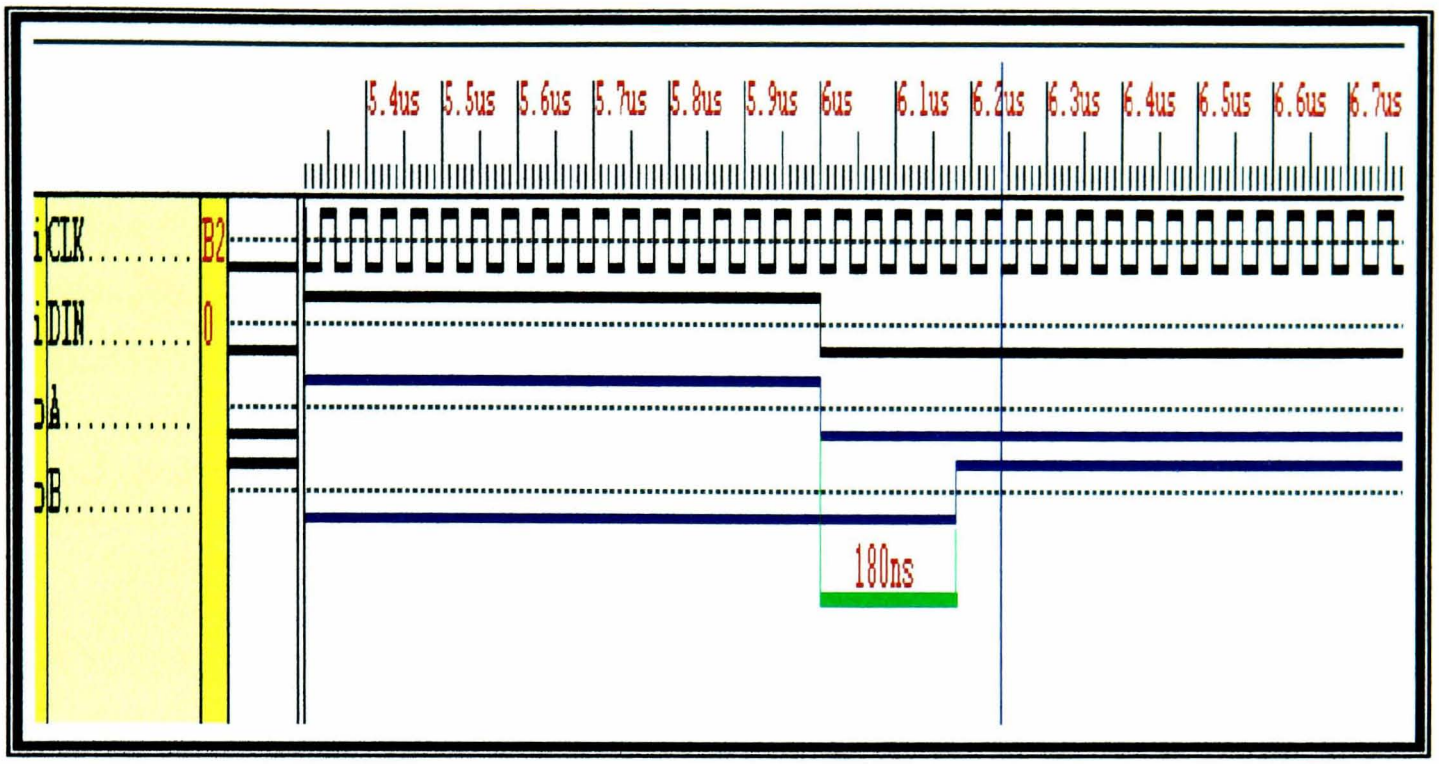


Figure 6.38 - *Simulated waveform shows the delay time*

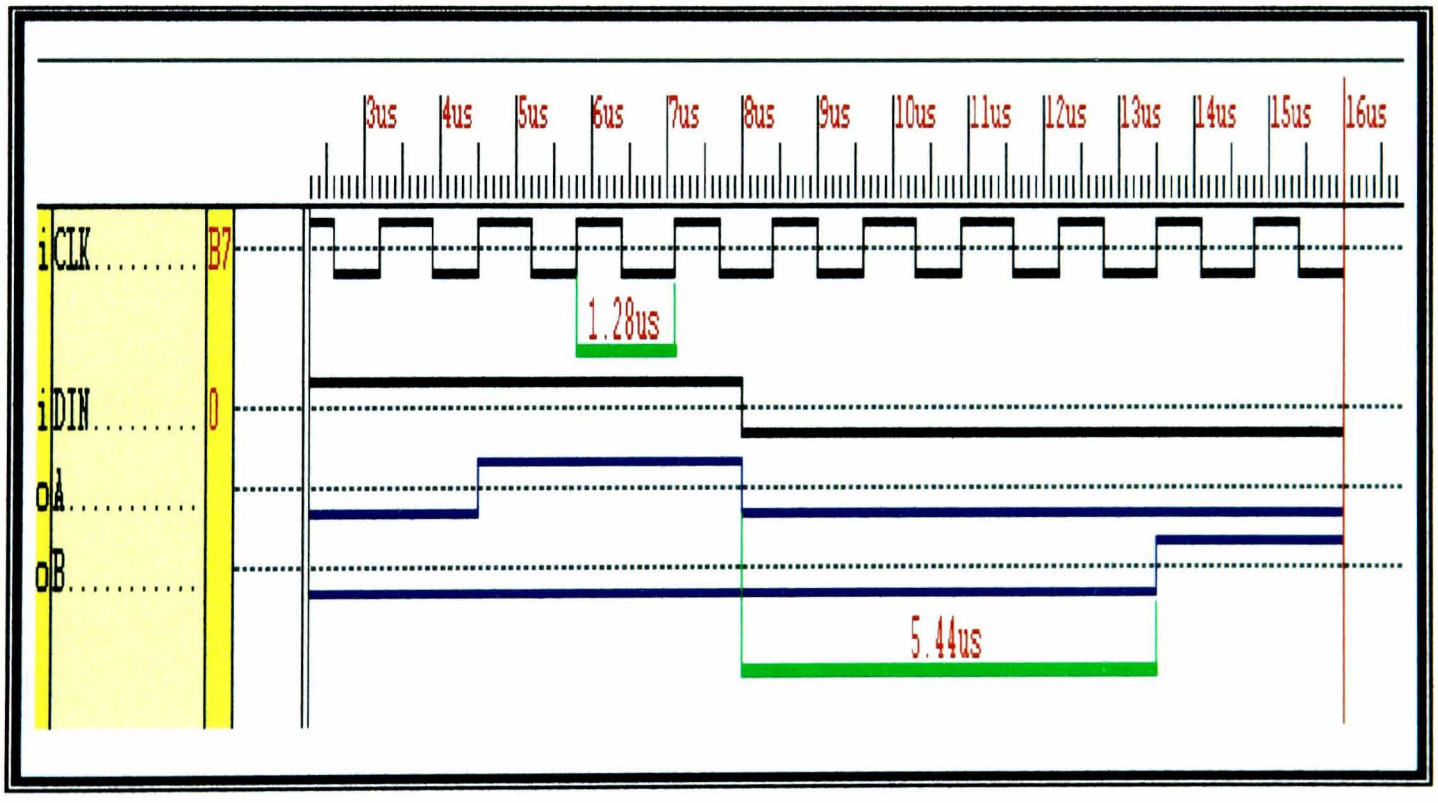


Figure 6.39 - *Simulated waveform showing the delay time*

6.9 The Overall Simulation of the Controller

All the components in Figure 6.1 can be tested at the same time using the testbench illustrated in Figure 6.40. For this task, a VHDL test component, ***testbench***, is created to simulate the overall control environment. The VHDL design of the testbench model is configured with the entity name ***testbench***.

```
ENTITY testbench IS
 PORT (clk:in std_logic;
       signal startt:out std_logic;
       signal ia,ib:out  STD_LOGIC_VECTOR (7 downto 0)
END testbench;
```

The entity is designed with one input signal *clk*. Three output signals provide simulated information on the phase current of the motor (i_a and i_b). These two currents are used as inputs to Figure 6.1. Whenever *Startt* is equal to the value of '1' a new value is assigned to i_a and i_b as illustrated in Figure 6.41.

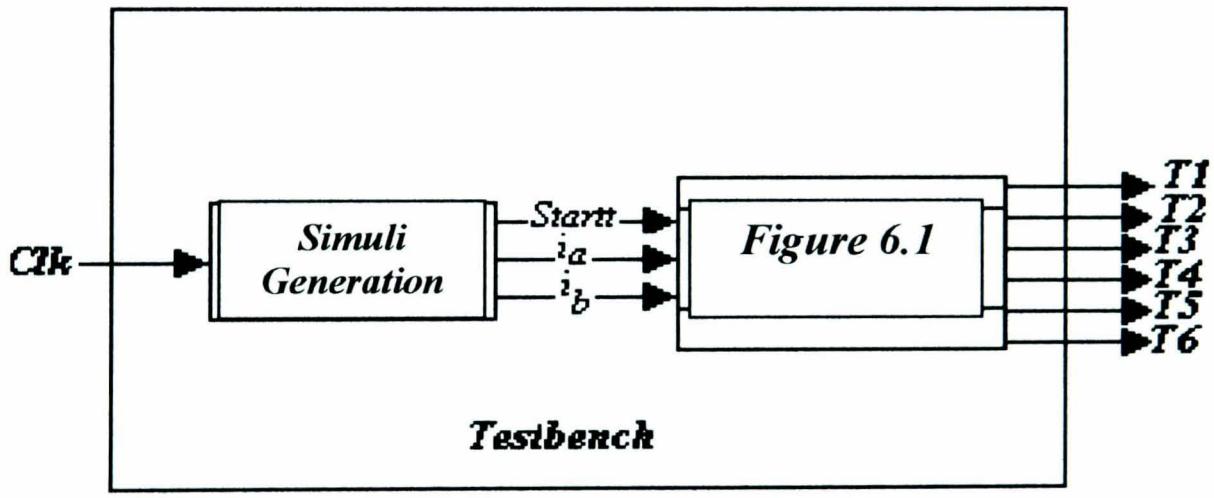


Figure 6.40 - *Model of a VHDL testbench*

The VHDL code automatically updates the values of the currents approximately every 100µs. This can be achieved in VHDL using a counter, represented by the following statement:

```
Signal counter: integer range Minmumvalue to Maxmvalue;
```

The associated architecture contains a single process that produces the data corresponding to the actual reading of the counter using a predefined constant of 26 std_logic_vector elements given in:

```
Archtecture testbench5_arch OF testbench5 IS
  signal counter: integer range 0 to 13000;
  signal counter1: integer range 0 to 1000;
  constant s1:STD_LOGIC_VECTOR (7 downto 0):="00000000";
  constant s2:STD_LOGIC_VECTOR (7 downto 0):="00010100";
  ... ... ... ... ... ... ... ... ... ... ...
  ... ... ... ... ... ... ... ... ... ... ... .
  ... ... ... ... ... ... ... ... ... ... ... .
  constant s13:STD_LOGIC_VECTOR (7 downto 0):="00000000";
_ ********* ***********
  constant c1:STD_LOGIC_VECTOR (7 downto 0):="00111100";
  constant c2:STD_LOGIC_VECTOR (7 downto 0):="00101000";
  ... ... ... ... ... ... ... ... ... ... ... ..
  ... ... ... ... ... ... ... ... ... ... ... ..
  ... ... ... ... ... ... ... ... ... ... ... ..
  constant c13:STD_LOGIC_VECTOR (7 downto 0):="00111100";
```

Since the algorithm is sequential, the entire operation is carried out in a VHDL Process.

```
process (clk,counter)
  begin
if clk'event and clk='1' then
      counter <=counter+1;
end if;
if counter <100 and counter >=0 then
      ia<=s1;
      ib<=c1;
      ... ... ... ... ... ... ... ... ... ...
      ... ... ... ... ... ... ... ... ... ...
elsif counter <1300 and counter >1200 then
      ia<=s13;
      ib<=c13;
elsif counter>1300 then
          counter<=0;
end if;
end process;
```

The complete listing of the VHDL code is included in *Appendix* C.

The overall simulation of the induction motor controller is illustrated in Figure 6.41, where i_a, i_b, i_{mref} and *ref_speed* are used as inputs and the switching patterns of the 6 power transistors *T1, T2, T3, T4, T5* and *T6* are the outputs of the FOC.

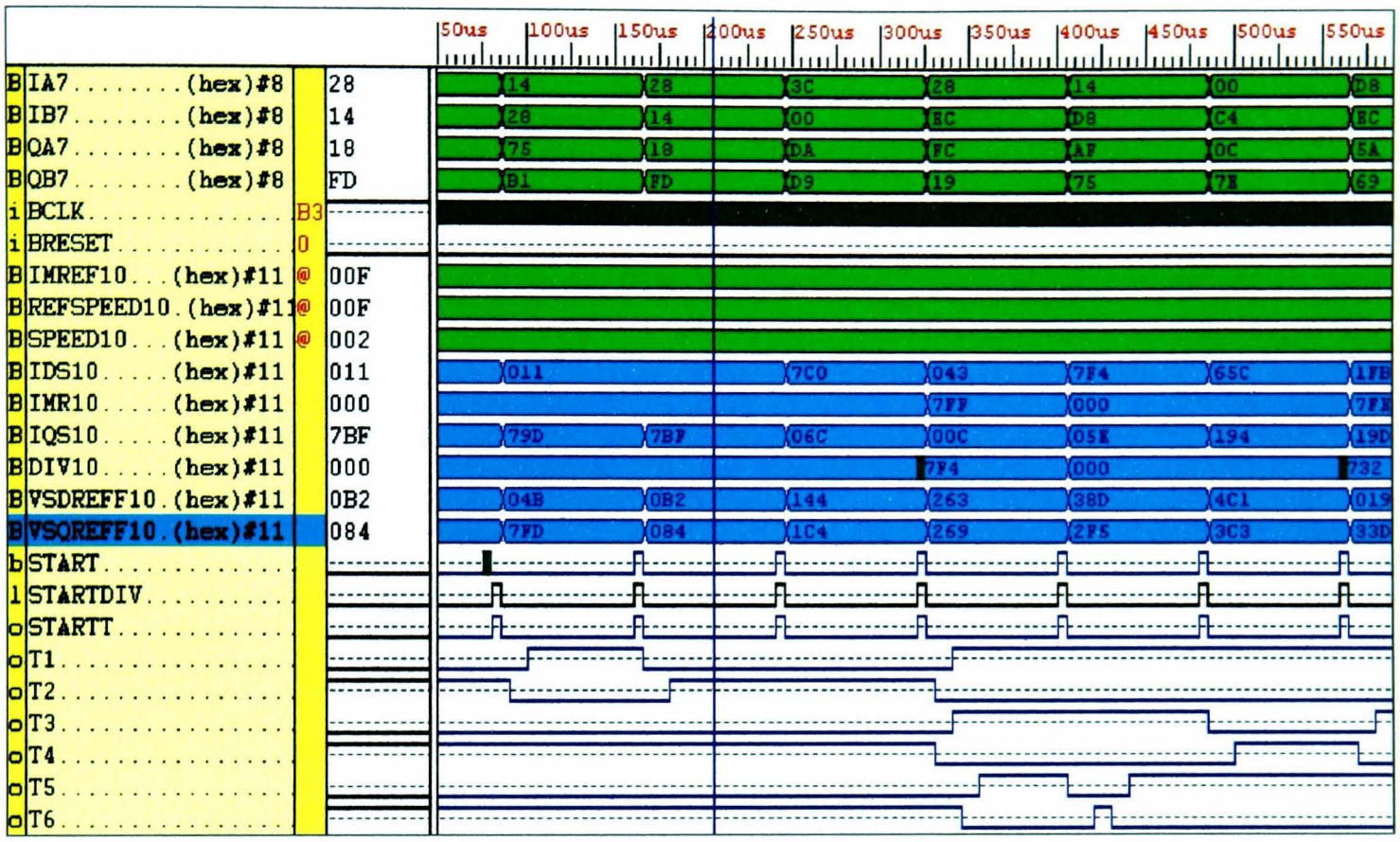


Figure 6.41 - *Overall simulation of the induction motor controller*

This chapter presented a new FPGA based control structure for a.c. drives and the development of a novel digital circuit realization scheme for induction motor vector control, employing a single FPGA Sparttan XCS40 from Xilinx, Inc. An important aspect in modelling the control system is the representation of the machine using the Clark and Park transforms, theta calculation, multiplication, division, PID controller and PWM generation. Their design in VHDL is fully described. The PWM waveform generator and performance characteristics are then demonstrated and the overall strategy is validated by simulation.

The main achievement of this original design solution consists of the universal compatibility of the VHDL designs (code) with respect to different EDA platforms, reusability of the VHDL code, compact and fast FPGA solution for implementation and high accuracy and speed of response of the controller.

In addition, the universal applicability of these control blocks to a wide range of vector control drives is an important asset. The experimental work presented in the next chapter seeks to validate the simulation results and therefore the controller design.

Chapter 7

Experimental tests

The overall objective of this chapter is to present the implementation and testing of the reusable modules for IM vector control. The first part includes experimental results relating to the performance of separate modules used for vector control. The second part shows the results related to the implementation of all the modules as one component on the FPGA, to illustrate the overall performance of the vector control system.

7.1 Hardware Design Process

Using the Xilinx Foundation F1.5 EDA tool [71], a VHDL design can be synthesised and then implemented into a particular device. Figure 7.1 shows a simplified block diagram of the process. In the synthesis stage, the VHDL code is converted into a netlist, which is a format that contains the hardware description of the design. The implementation is the process of converting the netlist into a bitstream file. The information in the bitstream file is used to configure the target FPGA. During the implementation process, the target technology and other hardware design specifications, such as pin location, has to be confirmed. For troubleshooting and analysis purpose, Xilinx Foundation also generates a status report at each stage of the process.

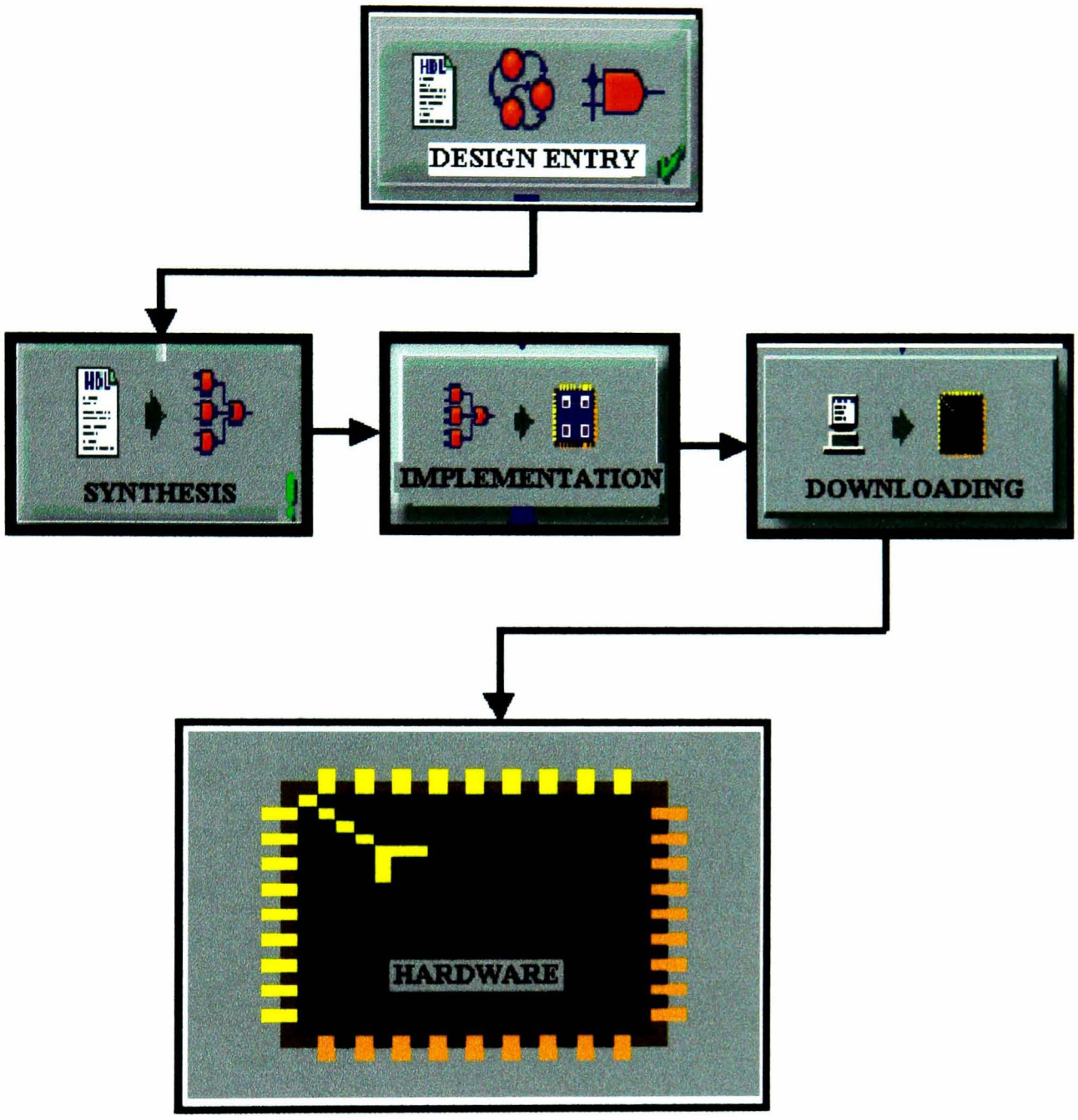


Figure 7.1 - *Simplified block diagram of hardware design process*

7.2 General Implementation Issues

The target technology for this design is the Spartan XCS40/XL, a part belonging to the Xilinx Spartan XCS**/XL series of FPGAs. These devices are fully reconfigurable and can be reprogrammed an unlimited number of times. The Spartan series is a family of high performance 3.3V or 5.0V devices based on SRAM technology. The features of this series are:

- System performance beyond 80 MHz.
- Density up to 1862 logic cells or 40000 system gates.
- Unlimited programmability.
- 3.3 V supply for low power with 5V tolerant I/Os.
- 12 mA or 24 mA output drive.

Table 7.1 shows other features of some of the devices in the family.

Device	**Logic Cells**	**Max system gate**	**Gate range**	**CLB Matrix**	CLB's	**No OF FlipF**	**Max user's I/O**
XCS05 & XCS05XL	238	5,000	2000-5000	10*10	100	360	77
XCS10 & XCS10XL	466	10000	3000-10000	14*14	196	616	112
XCS20 & XCS20XL	950	20000	7000-20000	20*20	400	1120	160
XCS30 & XCS30XL	1369	30000	10000-30000	24*24	576	1536	192
XCS40 & XCS40XL	1862	40000	13000-40000	28*28	784	2016	224

Table 7.1 - ***Features of Spartan and SpartanXL devices***

The design of the Vector Control Modules presented in previous chapters did not take into account the limitation of the target technology. These considerations have to be addressed when preparing a design for downloading into hardware FPGA or ASIC. In the present project, the design is prepared for implementation into Spartan XCS40PQ208 FPGA. This device has a total of 784 CLBs and an equivalent gate count of up to 40,000.

Figure 7.2 shows the hierarchy of individual subcomponents (subdesign entities) for the FOC, brought together in one higher level component (design entity).
This structure of the controller was developed following a hierarchical design and partitioning technique and relates to the implementation process required to convert the netlist into a format that can be downloaded into the target technology.

7.3 FPGA Implementation and Testing of the Vector Control System

When designing and modelling digital systems in VHDL, the design is partitioned into natural abstract blocks, known as components. Each component is the instantiation of a design entity, which is normally modelled in a separate system file for easy management and individual compilation by simulation or synthesis tools. The total system is then modelled using a hierarchy of components, known as "design hierarchy".
Each element of the Vector Control Modules is designed and carefully optimised for synthesis. Seven VHDL components are identified:
Clark_transform, ***Park_transform***, ***Current_model***, ***PI_controller***, ***Control_unit***, ***Inverse Park_transform*** *and* ***PWM generation***.

Each component consists of several subcomponents, which make up the core of the vector control algorithm. Figure 7.2 illustrates how these components relate to each other to form the complete control system.

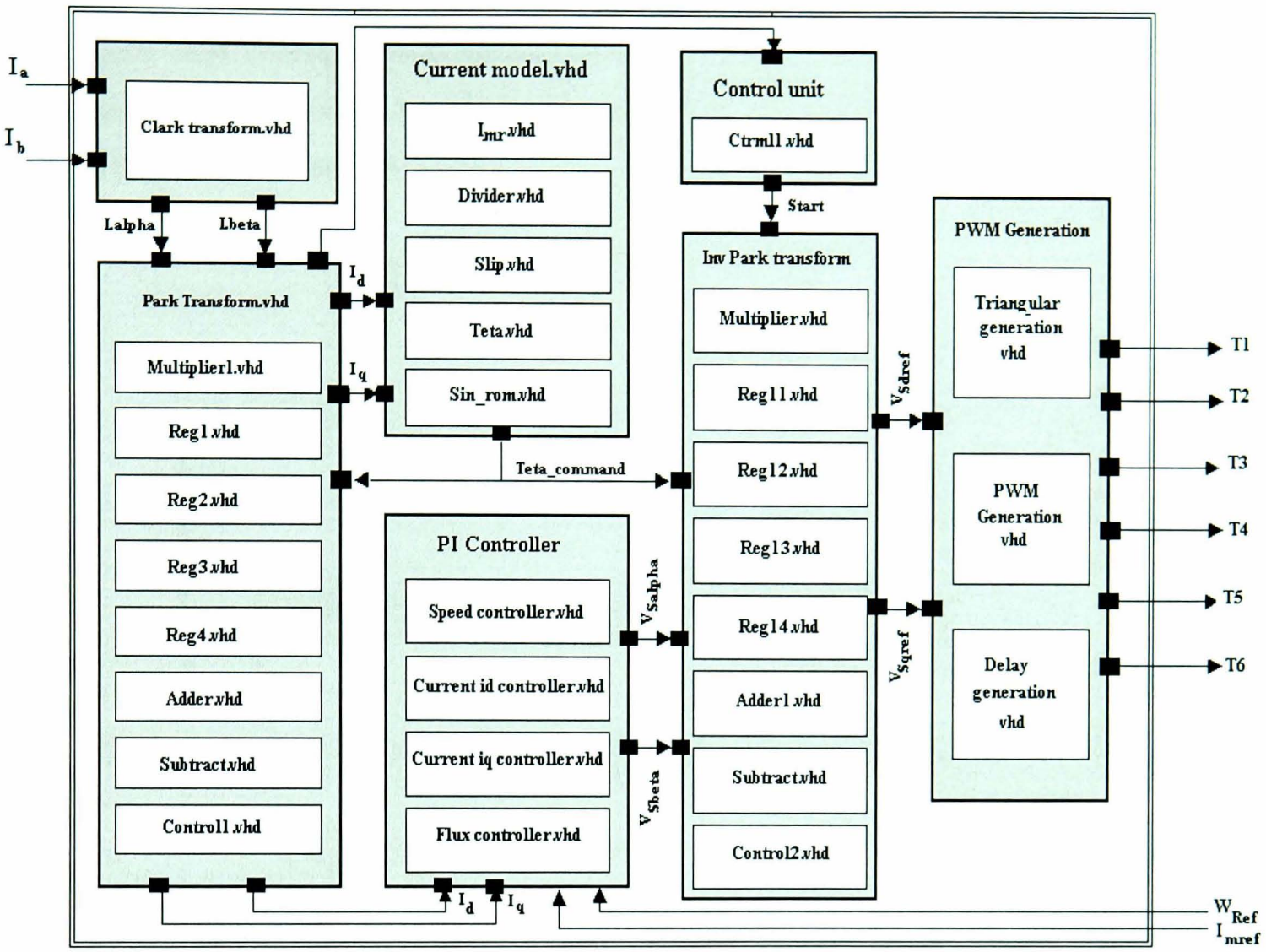


Figure 7.2 - *VHDL Hierarchy*

7.3.1 The Implementation of the Clark Transform

The Spartan XCS40 FPGA is available in several packages and the one used for this design is the PQ208 package, which has 224 I/O pins in total. Using the Xilinx Foundation Implementation tools, the design of the Clark transform component is compiled into a bitstream file. During the generation of the bit stream, the inputs and the outputs of the design are mapped to the physical I/O pins of the FPGA. The allocation of pin numbers can be either performed automatically by the implementation tools or explicitly specified by the user. In this design, all the input and output terminals are assigned to the physical I/O pins of the XCS40PQ208. The HDL editor was used to edit the Clark transform.ucf user-constraint file. Figure 7.3 shows the symbol of the Park transform component indicating the inputs/outputs of the module.

The Clark transform was first simulated, as illustrated by Figure 7.4 and Figure 7.6, then the results were compared with the practical ones, obtained after FPGA downloading and hardware testing, as shown by Figure 7.5 and Figure 7.7. The VHDL testbench in section 6.9 automatically updates the values of the currents *ia* and *ib* approximately every $100\mu s$.

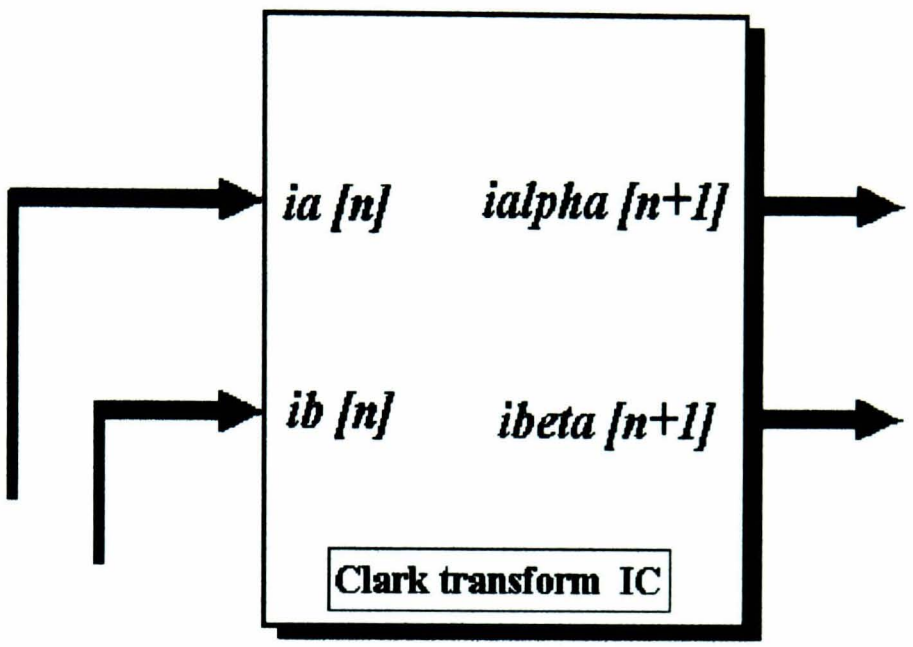


Figure 7.3 - *Symbol of the Clark transform component*

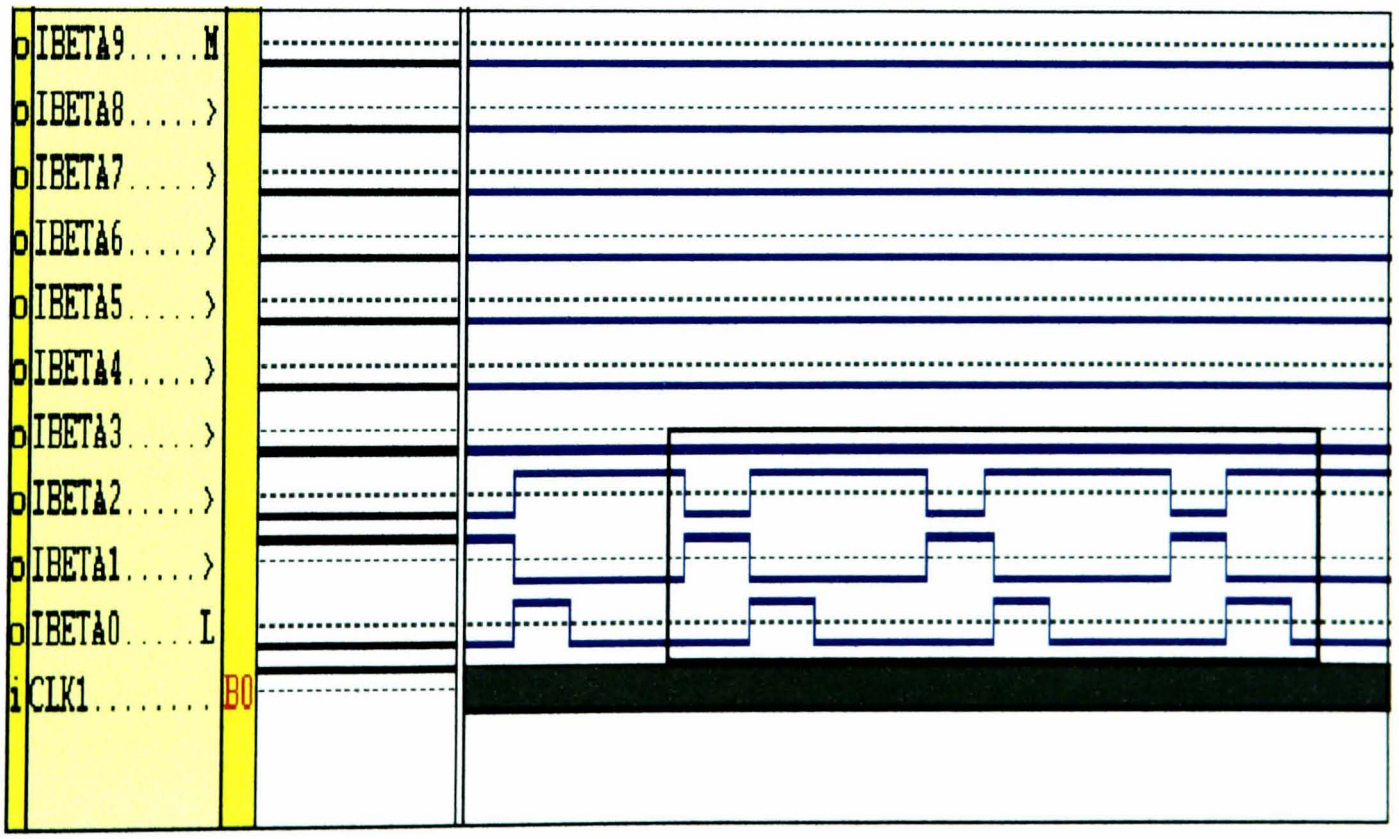


Figure 7.4 - *I-beta simulated waveform of Clark transform*

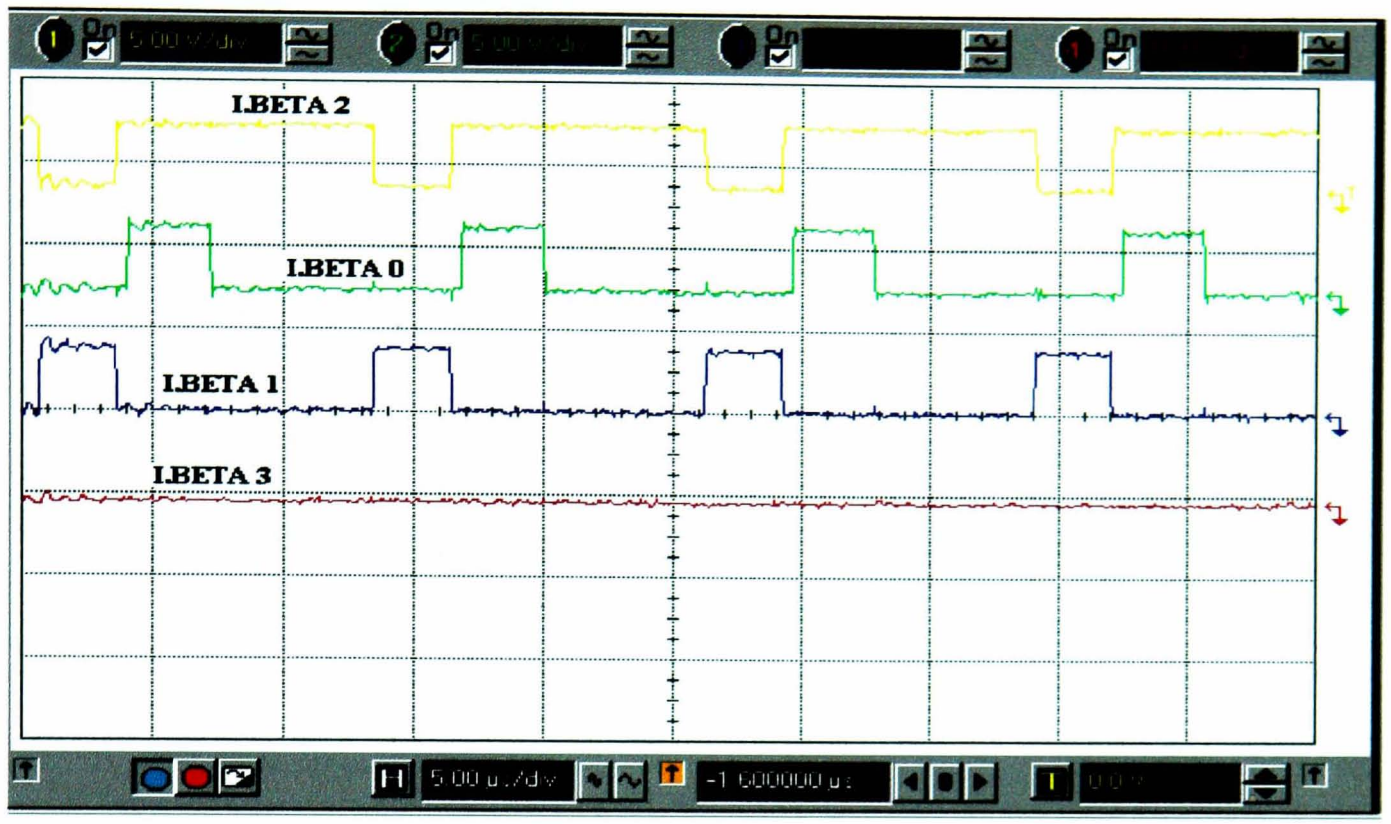


Figure 7.5 - *i-beta implemented waveform for Clark transform*

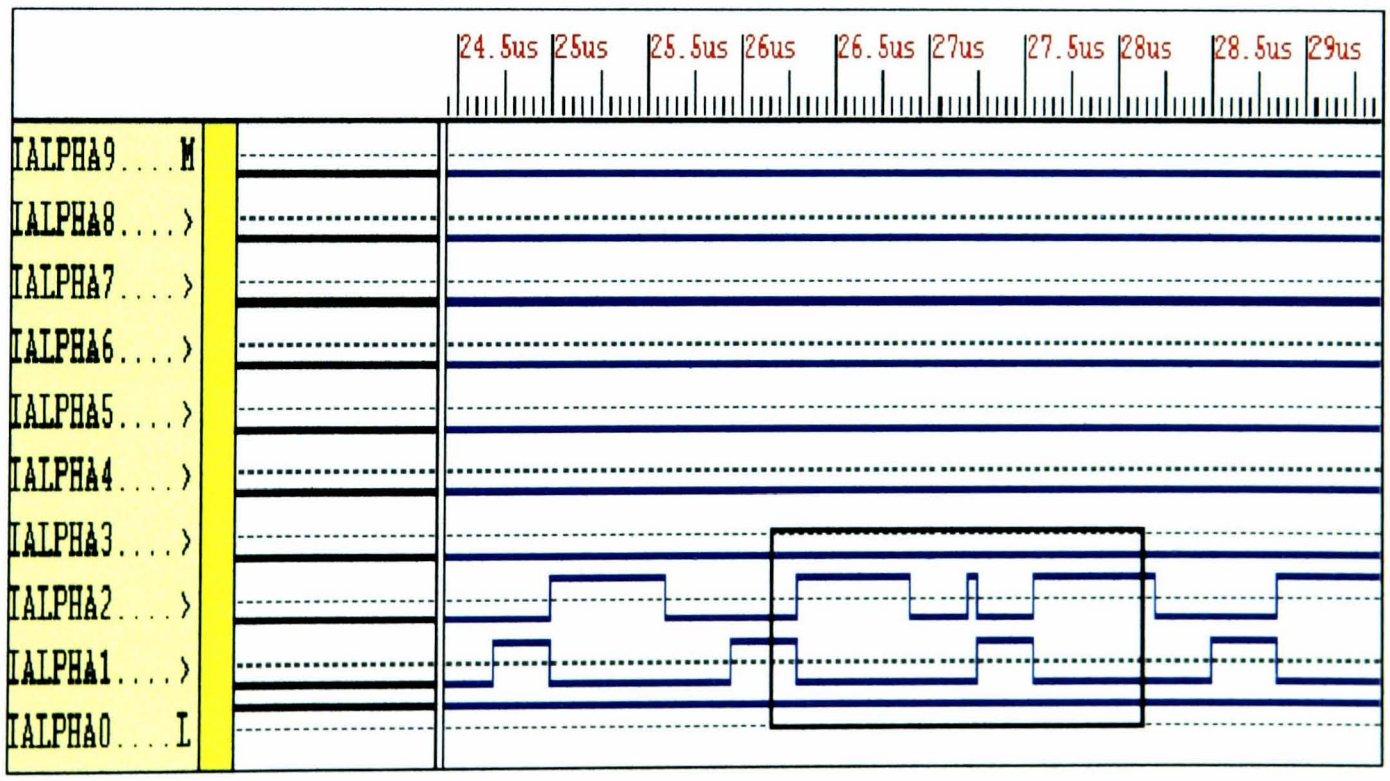


Figure 7.6 - *i-alpha simulated waveform of Clark transform*

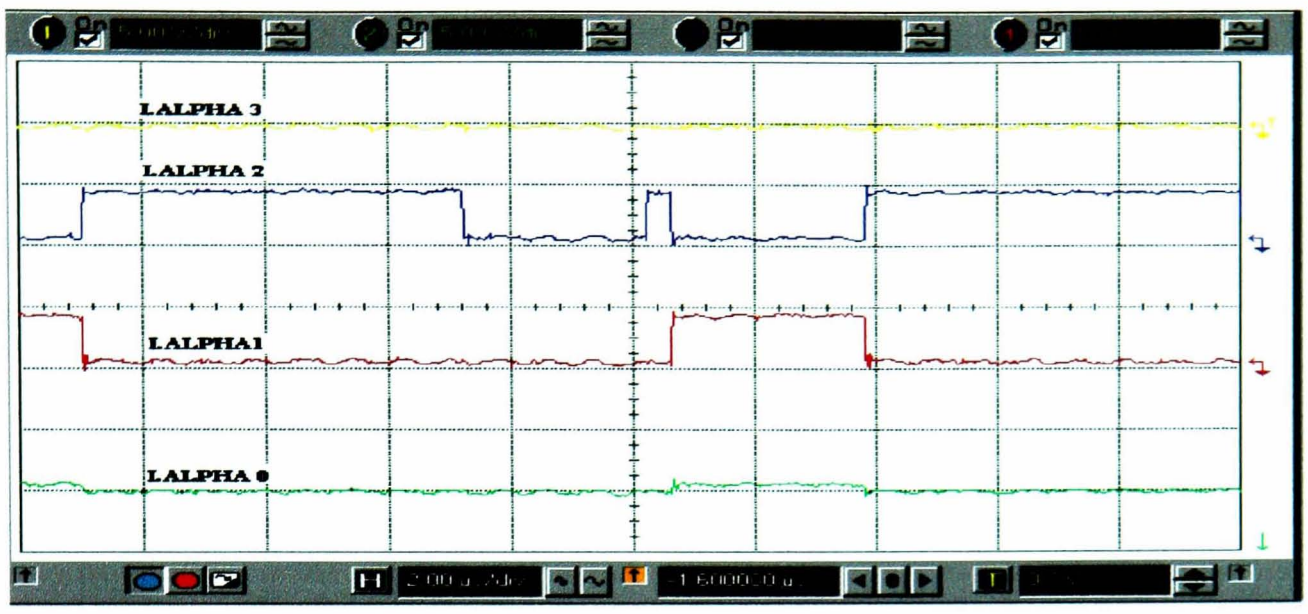


Figure 7.7 - *i-alpha implemented waveform for Clark transform*

Only the first four bits of i-alpha and i-beta are plotted, to be used for the comparison between the implemented designs and the simulated ones. It can be seen from the practical tests that the implemented values of i-alpha and i-beta for the first four LSBs has similar waveforms to the simulated values.

7.3.2 The Implementation of the Park Transform

Each element of the Park transform is designed and carefully optimised for synthesis. Eight subcomponents (multiplier, reg1, reg2, reg3, reg4, adder, subtracter and control1) make up the core of the Park transform, as illustrated in Figure 7.2. Figure 7.8 shows the symbol of the Park transform component, indicating the inputs/outputs of the module.

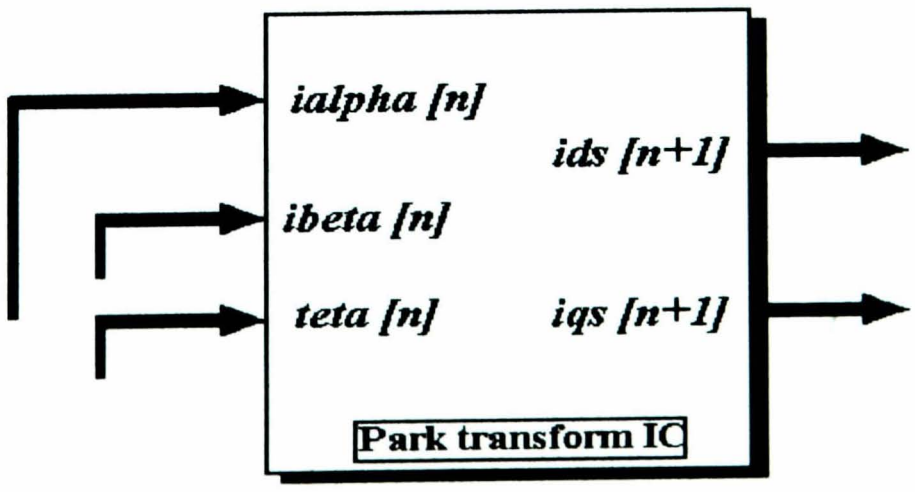


Figure 7.8 - *Symbol of the component Park transform*

The nature of the component connection and the functionality of the process are described in chapter 6 section 2. In this design, all the input and output pins are specified manually by assigning the appropriate pin numbers to the input and output of the Spartan XCS40PQ208. Before compiling the design, the following constraints were placed in the Park transform.ucf file:

NET CLOCK	LOC = P2;	NET IDS(7)	LOC = P10;
NET RESET	LOC = P3;	NET IDS(6)	LOC = P11;
NET IDS(11)	LOC = P4;	NET IDS(5)	LOC = P12;
NET IDS(10)	LOC = P5;	NET IDS(4)	LOC = P14;
NET IDS(9)	LOC = P8;	NET IDS(3)	LOC = P15;
NET IDS(8)	LOC = P9;	NET IDS(2)	LOC = P20;
NET IDS(1)	LOC = P21;	NET IDS(0)	LOC = P22;

Table 7.2 - ***Pin location for Sparttan XCS40***

It can be seen that the clock input of the design is allocated to pin 2, the reset to pin 3 and so on. The netlist is compiled into a bit stream file using the implementation procedure in Xilinx Foundation Project Manager.

```
Number of CLBs:             262 out of  784   33%
    CLB Flip Flops:         195
    4 input LUTs            438
    3 input LUTs:           29 (12 used as route-throughs)
Number of bonded IOBs:      62 out of  169   36%
    IOB Flops:              0
    IOB Latches:            0
Number of clock IOB pads:   1 out of   8   12%
Number of primary CLKs:     1 out of   4   25%
Number of secondary CLKs:   3 out of   4   75%
Number of RPM macros:       18
Total equivalent gate count for design: 5547
```

Figure 7.9 - *Synthesis report for the Clark and Park transform*

The following points can be noted from the synthesized report of the Clark and Park transforms:

- The number of bounded I/Os in the device is sufficient for the design.
- The total equivalent gate count is below the maximum count stated in the data book.
- The Clark and Park transforms combined together require 262 CLBs which is below the maximum allowed value.

From these observations it can be seen that the design has a total count of 5547 gates which is considerably less than the maximum device limit.
Figures 7.10, 7.11, 7.12 and 7.13 show the results of the simulated and the implemented waveforms respectively for the Clark and Park transforms combined together. The practical test results show that the implemented waveforms confirm the results obtained through simulation.

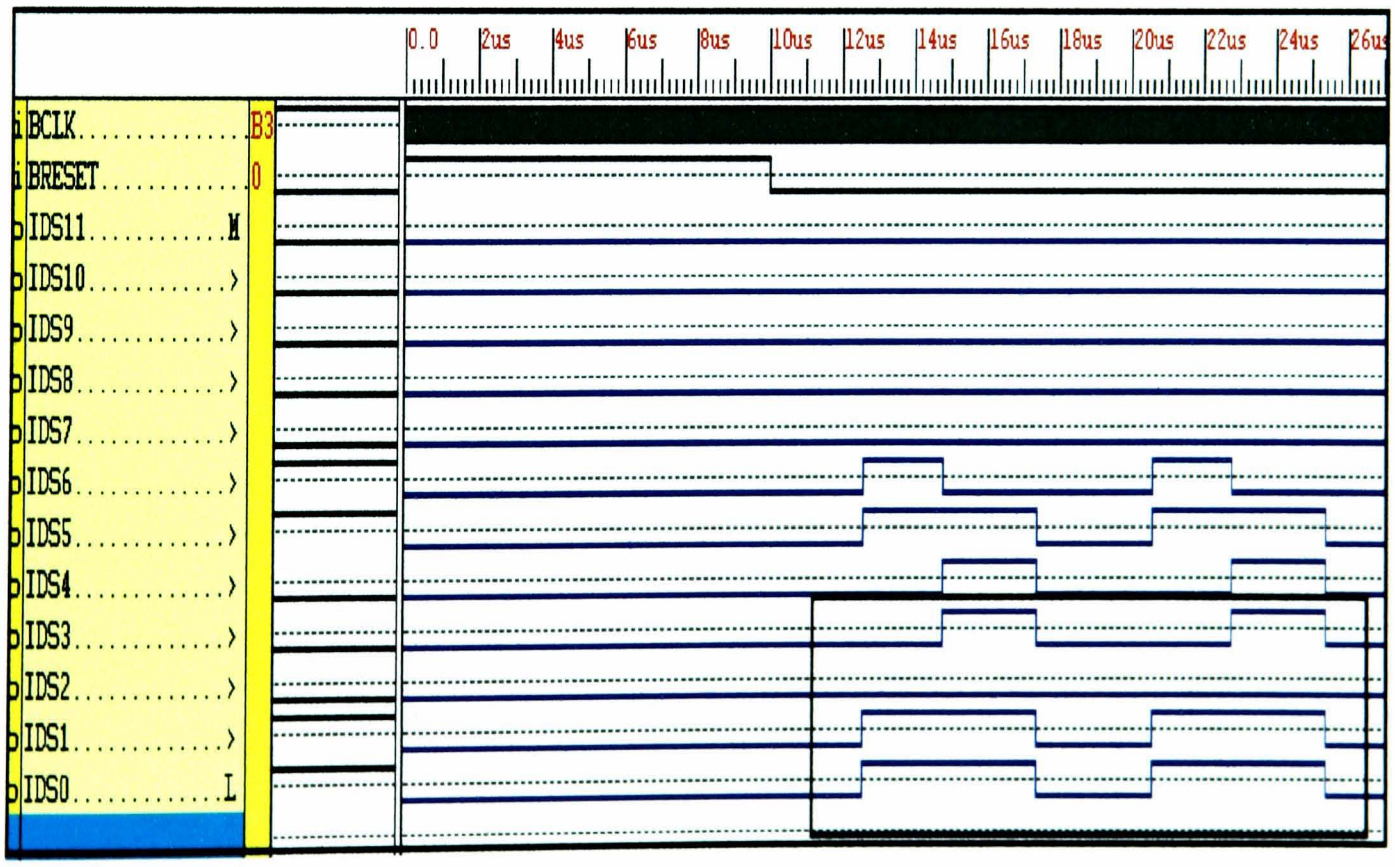


Figure 7.10 - *Waveforms of i_d for the simulated circuit of the Park transform.*

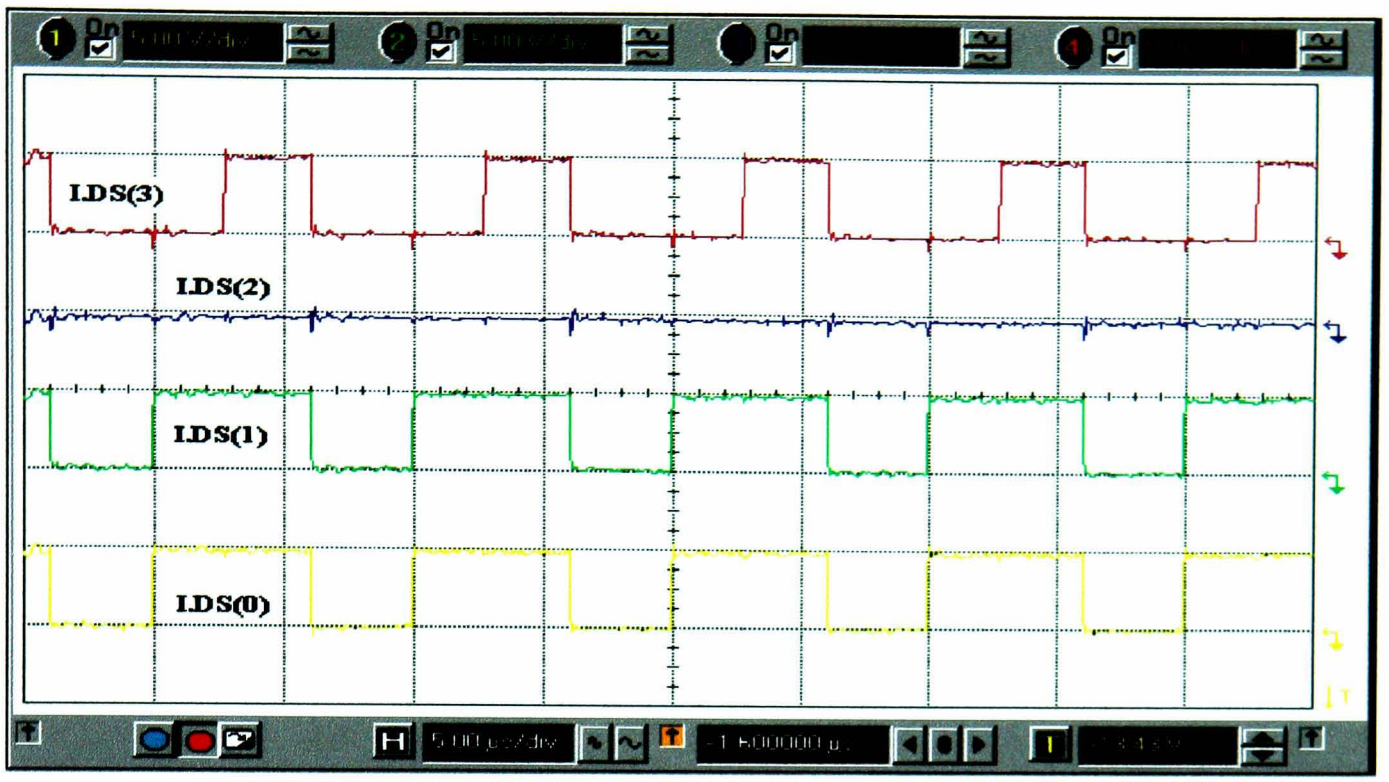


Figure 7.11 - *Waveforms of i_d for the implemented circuit of the Park transform.*

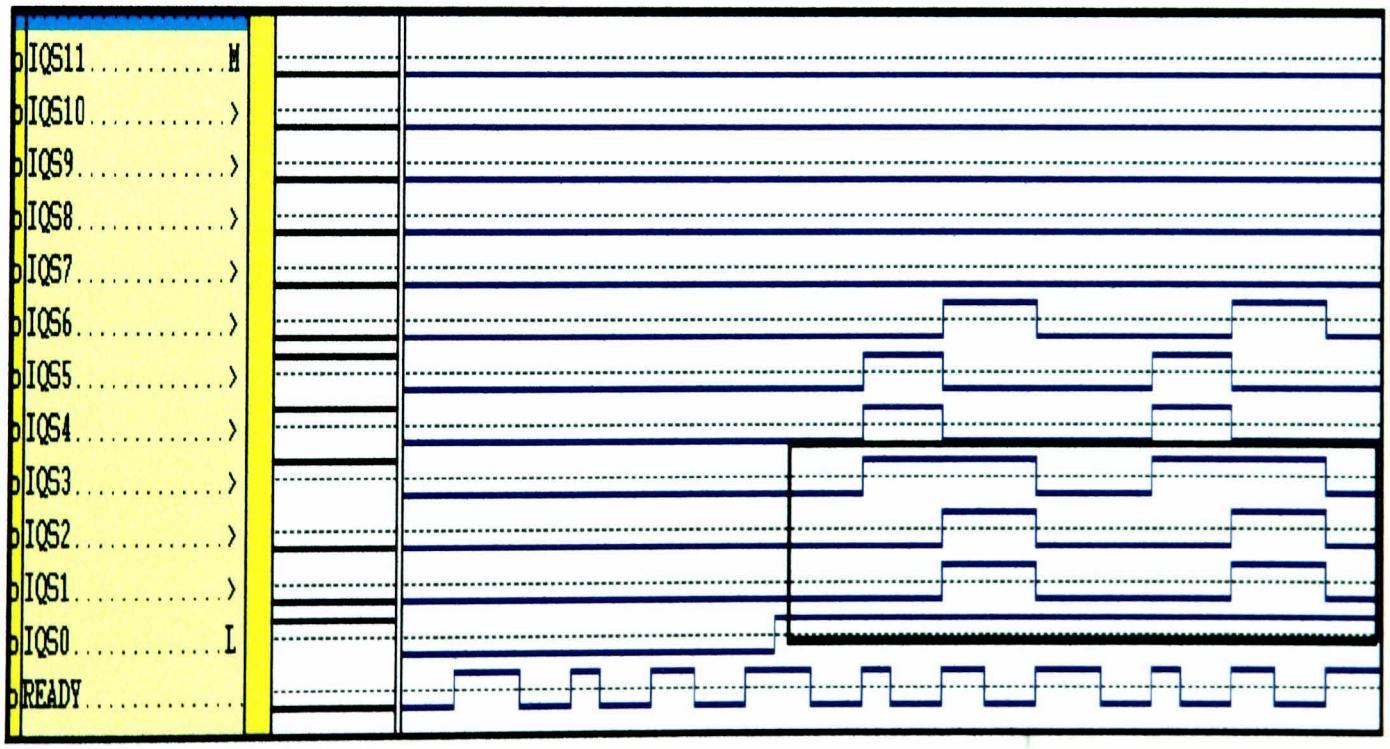


Figure 7.12 - *Waveform of i_q for the simulated circuit of the Park transform.*

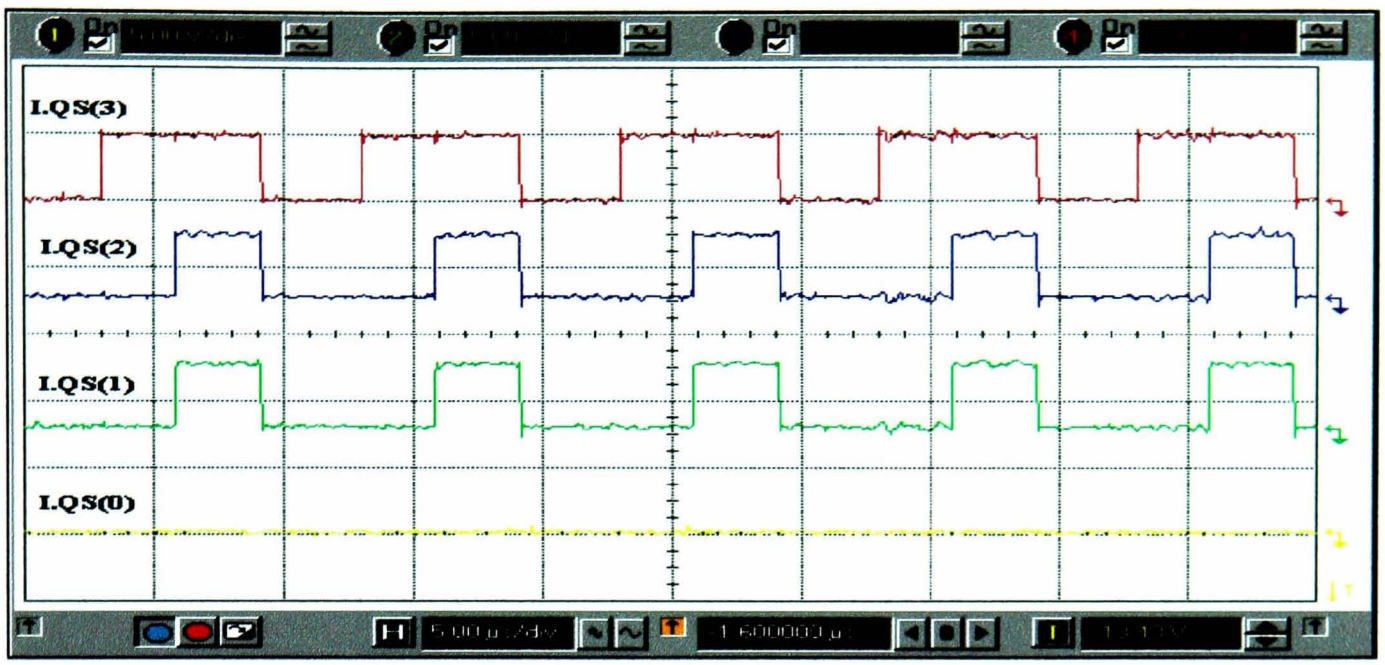


Figure 7.13 - *Waveform of i_q for the implemented circuit of the Park transform*

Figure 7.11 and Figure 7.13 show the experimental results of the test performed by downloading the Park transform component design in the Xilinx FPGA. Only the first four bits of i_d and i_q are plotted to be used for the comparison between the implemented design and the simulated ones of Figure 7.10 and Figure 7.12. From the practical tests it can be seen that the implemented design generates the same values of i_d and i_q for the first four LSBs as the simulated values highlighted in the rectangular box.

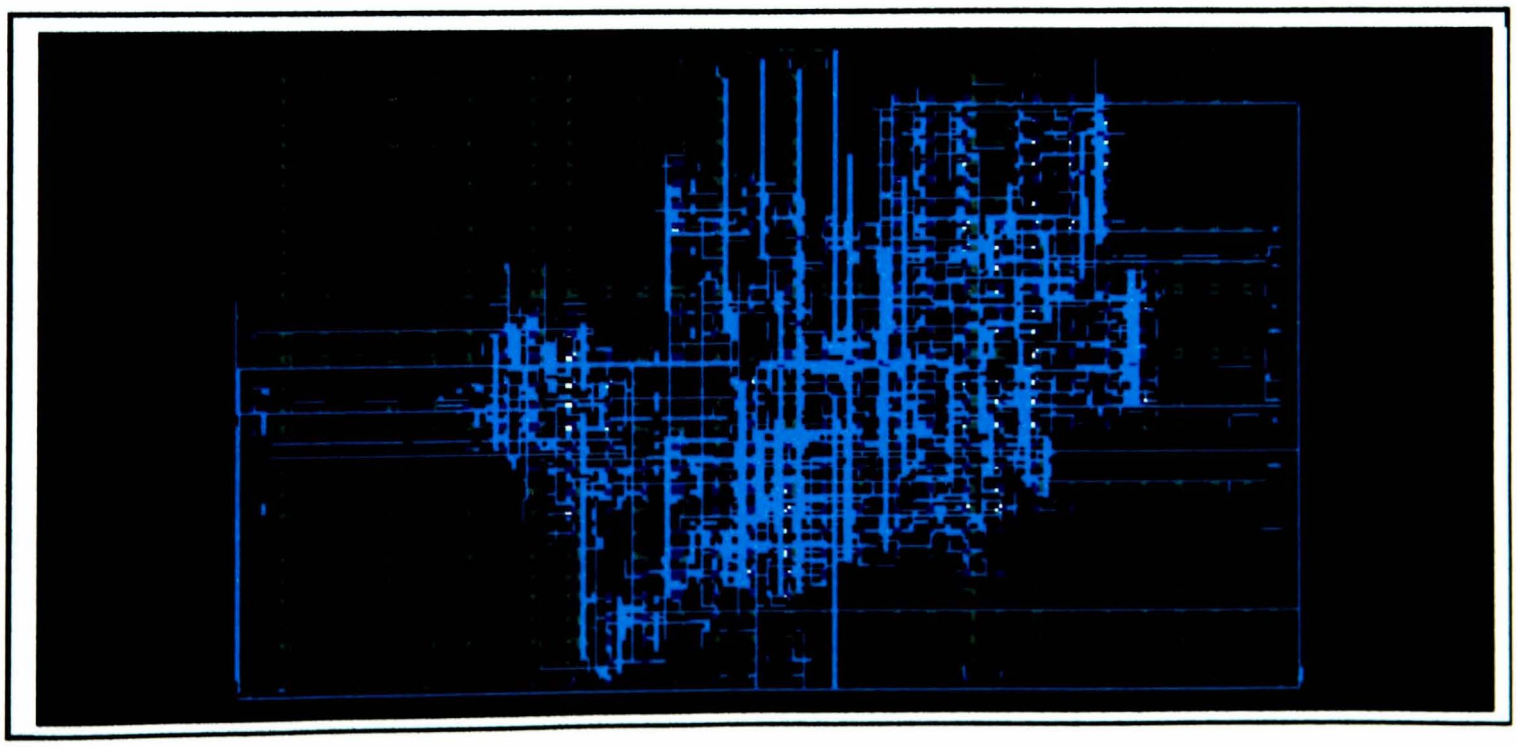


Figure 7.14 - *FPGA layout of Clark and Park transform*

A partial layout of the Clark and Park transforms synthesised into a Spartan S40PQ208 Xilinx FPGA is illustrated in Figure 7.14.

7.3.3 The Implementation of the Multiplier

Signal processing applications require fast and efficient digital multipliers. The efficiency of a multiplier can be assessed from the speed standpoint and from the chip area point of view. Digital multipliers are needed in many system applications, including digital filters, correlators and neural networks. These multipliers are typically required to handle operands of up to 16 bits and need to provide results in less than 50ns (20 MHz systems).
The most compact multipliers have a sequential operation decomposing the multiplication AxB into a set of additions and shifting operations. At each calculation cycle, an operand A is multiplied with the least significant bit of B and the result is added into the main shift register after which both registers are shifted with one position to the right. The one bit multiplication can be implemented with a set of 2 input AND gates. If the bit belonging to B is 0, the AND gates outputs are all zero. If the bit from operand B is 1 then the AND gates outputs repeat the bits of A.

A mixed architecture compromising between the size of sequential multipliers and the speed of matrix multipliers is presented in [72]. The basic principle is similar to the algorithm used by sequential multipliers: it consists of decomposing the multiplication of operands A and B into a series of simpler calculation cycles. Each operation cycle consists in multiplying the operand A with the least significant N_{sl} bits of B and adding the result into a shift register; both the result and operand B are then shifted with N_{sl} positions to the right. The N_{sl} number is called the multiplier step length. If the step length equals the operand B length then the multiplier works in the manner of a matrix multiplier.

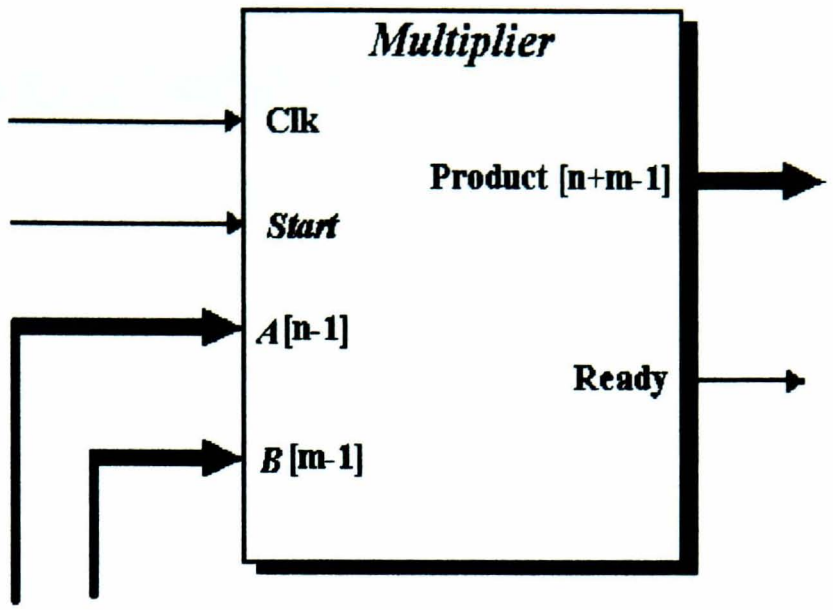


Figure 7.15 - *Symbol of the component multiplier*

Figure 7.15 shows the symbol of the component multiplier, clearly indicating the inputs/outputs of the module. The scheme used for implementing the test procedure is presented in Figure 7.16. It consists of: ADC0804, Spartan XCS40PQ208 FPGA, I/O digital card and PC. The implemented model consists of an 8-bit by 7-bit multiplier. This module is actually a part of the vector control scheme and is used extensively in the digital design of the FOC. The input to this module, consisting of operands A and B, is shown in Figure 7.17. Implementation results are shown in table 7.3.

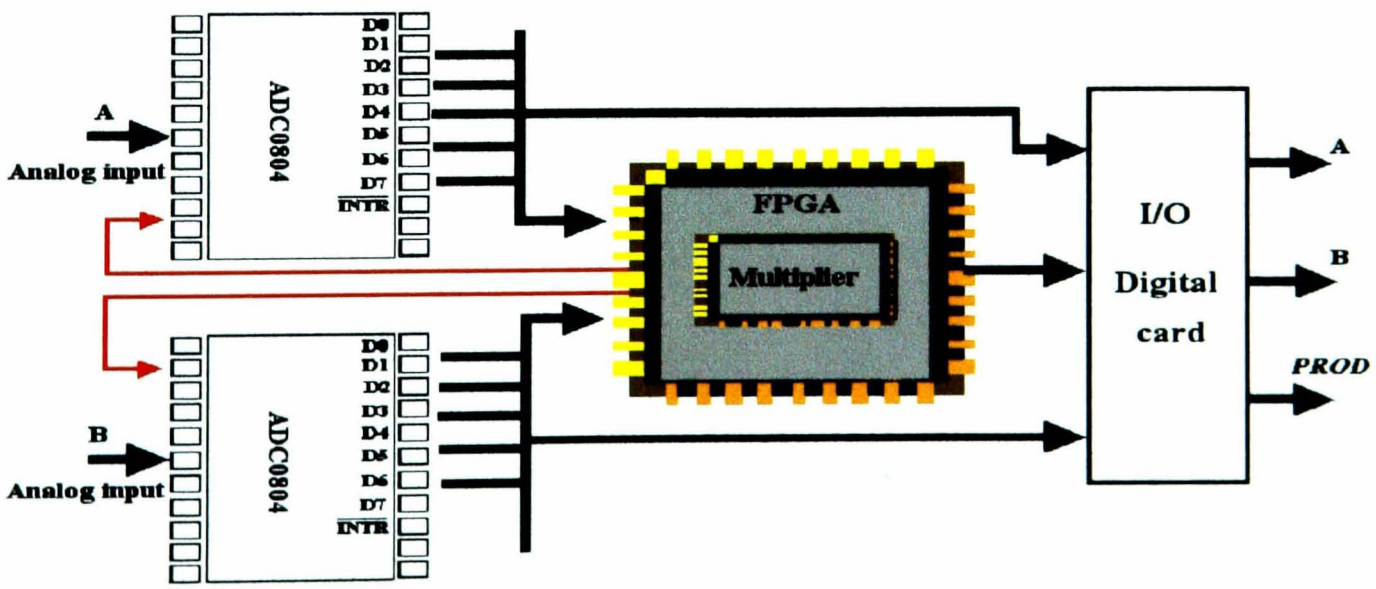


Figure 7.16 - ***Illustration of the implemented multiplier construction***

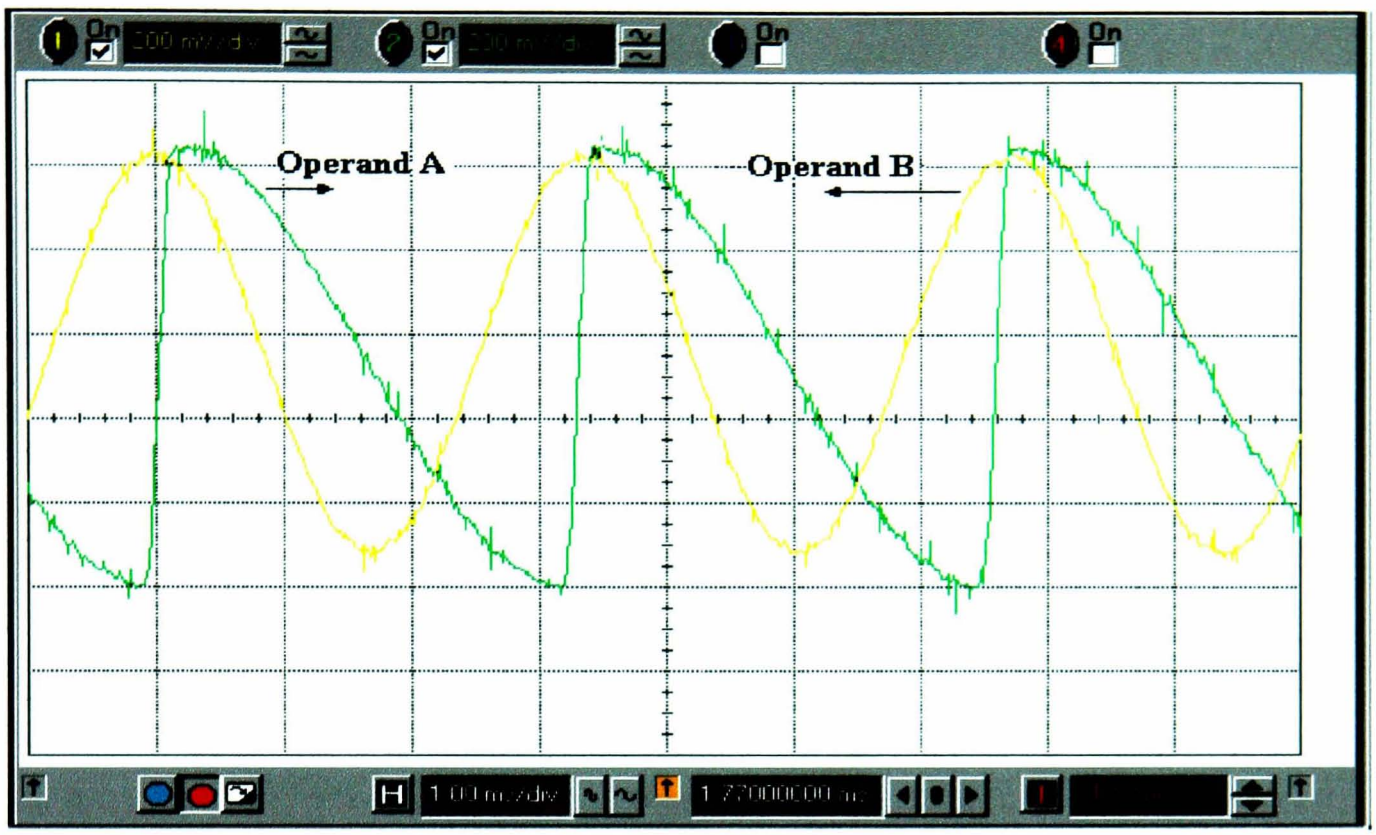


Figure 7.17 - *The inputs used for implementing the multiplier*

Figure 7.18 shows an extract from the implementation status report of the multiplier design which includes the clocking circuit designed to drive the two ADCs. The implementation is targeted at the Xilinx Spartan XCS40PQ208 FPGA. The following points can be noted from the report:

- The design requires 67 CLB's, which is a lot less than the number readily available in a Xilinx chip.
- The total equivalent gate count for the design is below the maximum count stated in the data book.
- The number of bounded IOBs in the device is sufficient for the design.

Time	A	B	Multiplication
0.000	15.000	5.000	75.000
0.000	3.000	8.000	24.000
0.000	0.000	12.000	0.000
0.000	19.000	6.000	114.000
0.001	0.000	16.000	0.000
0.002	0.000	20.000	0.000
0.002	10.000	15.000	150.000
0.002	6.000	9.000	54.000
0.002	0.000	7.000	0.000
0.002	7.000	16.000	112.000
0.002	25.000	10.000	250.000
0.002	0.000	20.000	0.000
0.003	8.000	20.000	160.000
0.003	26.000	9.000	234.000
0.003	0.000	8.000	0.000
0.003	10.000	5.000	50.000
0.004	18.00	9.000	162.000
0.004	12.000	16.000	192.000
0.004	0.000	24.000	0.000
0.005	8.000	5.000	40.000
0.005	0.000	10.000	0.000
0.005	14.000	11.000	154.000
0.006	1.000	9.000	9.000
0.006	4.000	27.000	108.000
0.006	28.000	5.000	140.000

Table 7.3 – The implementation results

```
Design Summary
--------------
   Number of errors:          0
   Number of warnings:        0
   Number of CLBs:                63 out of
784
      CLB Flip Flops:        42
      4 input LUTs:         112
      3 input LUTs:          17
   Number of bonded IOBs:         28 out of
169
      IOB Flops:              1
      IOB Latches:            0
   Number of clock IOB pads:      1 out of
8
   Number of primary CLKs:        1 out of
```

Figure 7.18 – Synthesis report for top hierarchy entity: multiplier

7.3.4 The Implementation of the Divider

Figure 7.19 shows the symbol of the component divider, clearly indicates the inputs/outputs of the module. The implemented model consists of an 8-bit by 8-bit divider. This module is used to divide the current in the q-axis (i_{qs}) by the magnetizing current (i_m). If the division is by zero then the answer is zero. Implementation result are shown in table 7.4. The experimental set up of the divider implementation, illustrated by Figure 7.20, consists of two ADC0804, a Spartan XCS40PQ208 FPGA, an I/O digital card and a PC.

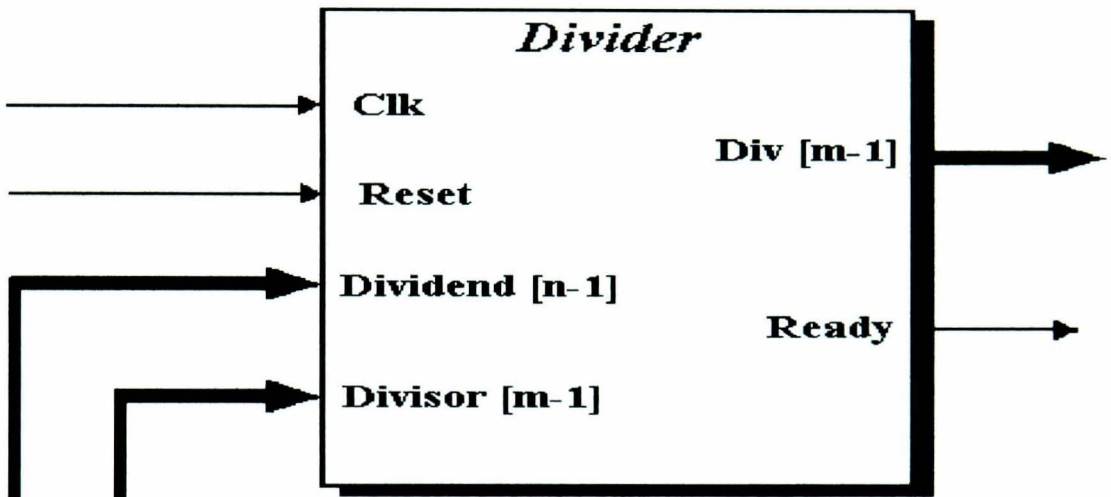


Figure 7.19 - Symbol of the component divider

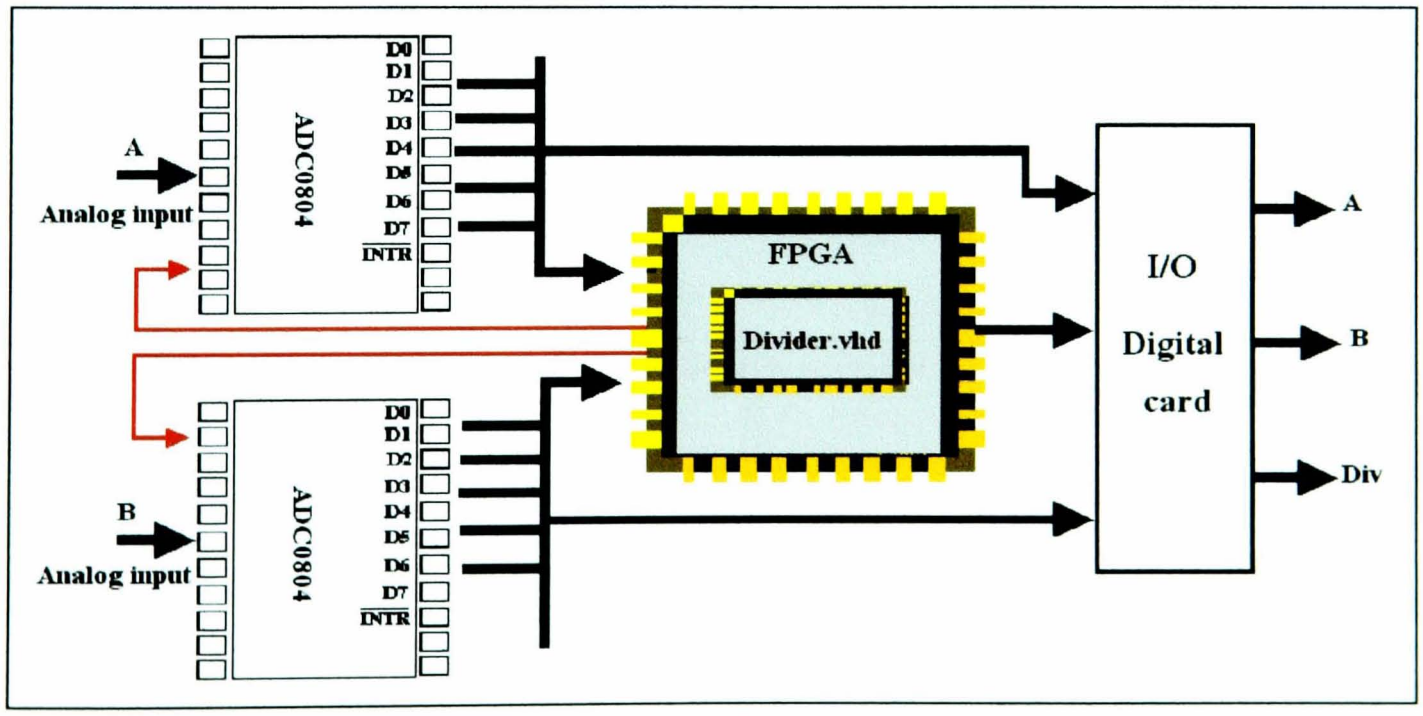


Figure 7.20 – illustration of the construction of the implemented divider

Another important issue to consider, particularly regarding implementation, is the area optimization. Figure 7.21 shows an extract from the implementation status report of the divider design which includes the clocking circuit designed to drive the two ADC: targeted to Xilinx Spartan XCS40PQ208 FPGA implementation. The following points can be noted from the report:

- The design requires 67 CLB's, a lot less than is readily available in a Xilinx chip.
- The total equivalent gate count for the design is below the maximum count stated in the data book.
- The number of bounded IOBs in the device is sufficient for the design.

Time	*A*	*B*	*Div*
0.000	2.480	2.383	1.000
0.001	2.480	2.344	1.000
0.002	2.480	2.148	1.000
0.003	2.480	1.094	2.000
0.004	2.480	0.645	3.000
0.009	2.480	0.059	42.000
0.009	2.480	0.020	126.000
0.009	2.480	0.000	0.000
0.009	2.480	0.020	0.000
0.016	2.480	-0.313	249.000
0.016	2.480	-0.352	249.000
0.016	2.480	-0.391	250.000
0.017	2.480	-0.762	253.000
0.021	2.480	-2.305	255.000
0.021	2.480	-2.305	255.000
0.031	-2.500	-2.305	1.000
0.031	-2.500	-2.305	1.000
0.031	-2.500	-2.305	1.000

Table 7.4 – The implementation results

```
Design Summary
--------------
   Number of errors:          0
   Number of warnings:        4
   Number of CLBs:                67 out of    784     8%
      CLB Flip Flops:         46
      4 input LUTs:          119
      3 input LUTs:           13
   Number of bonded IOBs:         38 out of    169    22%
      IOB Flops:               9
      IOB Latches:             0
   Number of clock IOB pads:      1 out of      8    12%
   Number of primary CLKs:        1 out of      4    25%
   Number of secondary CLKs:      3 out of      4    75%
   Number of RPM macros:          4
Total equivalent gate count for design: 1326
Additional JTAG gate count for IOBs:    1824
```

Figure 7.21 - *Synthesis report for top hierarchy entity: Clocking-divider5*

7.3.5 The PWM Control Signals Output Block

PWM d.c.-a.c. converters may serve a wide range of applications in a.c. motor drives and a.c. power conditioning systems. The PWM strategy plays an important role in the minimization of harmonics and switching losses in these converters, especially in three-phase applications. The Xilinx Spartan XCS40PQ208 FPGA is selected for implementing of the proposed PWM module.

The PWM VHDL model design is divided into two processes. One is used to describe the operational logic of the triangular waveform generator process unit, whereas the other process is used to generate the correct signal to control the transistors in the PWM inverter. The output signals defining the desired output voltage of the PWM inverter are transformed into six logic signals. Simulation results are fully analysed in Chapter 6, Section 6.8.

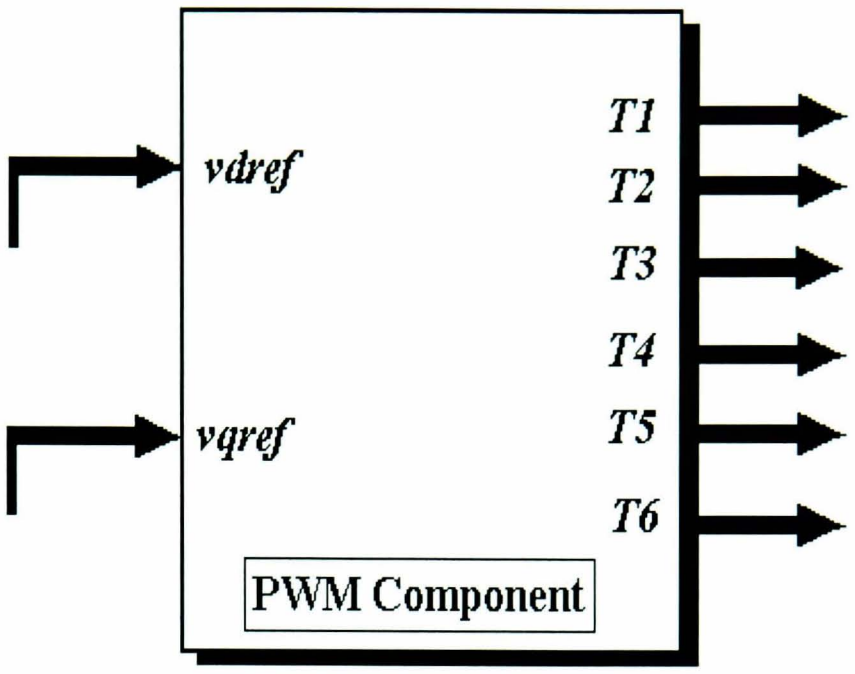


Figure 7.22 - Symbol of the PWM component

Figure 7.22 shows the symbol of the PWM component, clearly indicating the inputs/outputs of the module. Figure 7.23 and Figure 7.24 illustrate the experimental results of the PWM module operation. These results are obtained by testing the controller using an emulator written in VHDL. The emulator supplies the phase current values to the overall controller every 100μs, therefore allowing to carry out the experiment without the need for the motor or the interface circuit to be connected. Once the controller is working properly, the motor and the interface circuit are connected, knowing that it will work first time, without damaging any of the interface circuit or the motor, as shown in section 7.4.2.

Each signal in Figure 7.23 is amplified and applied to one power transistor. The edges of these signals are shifted by approximately 5 μs so that short circuits are avoided between transistors on the same inverter phase (*T1* and *T2* for example). Figure 7.23 presents four of the PWM control signals generated by the FPGA motor controller. They have been monitored using a four channel Hewlett Packard digital oscilloscope. The figure demonstrates the correlation between two of the signals that control the transistors on the same inverter leg *T1* and *T2*. The two signals have complementary values: when one of them is '0' the other is '1', as illustrated in Figure 7.24.

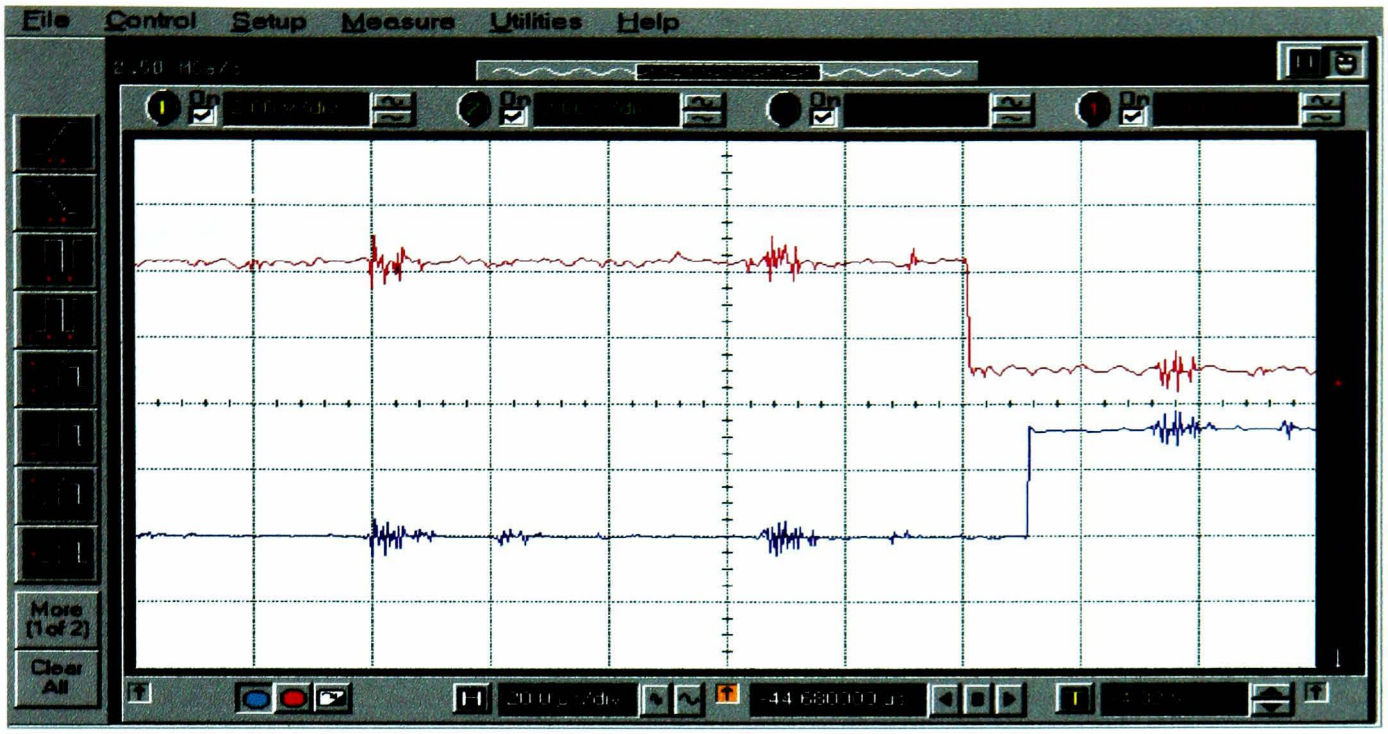


Figure 7.23 - *Four PWM output signals controlling the transistors in the inverter*

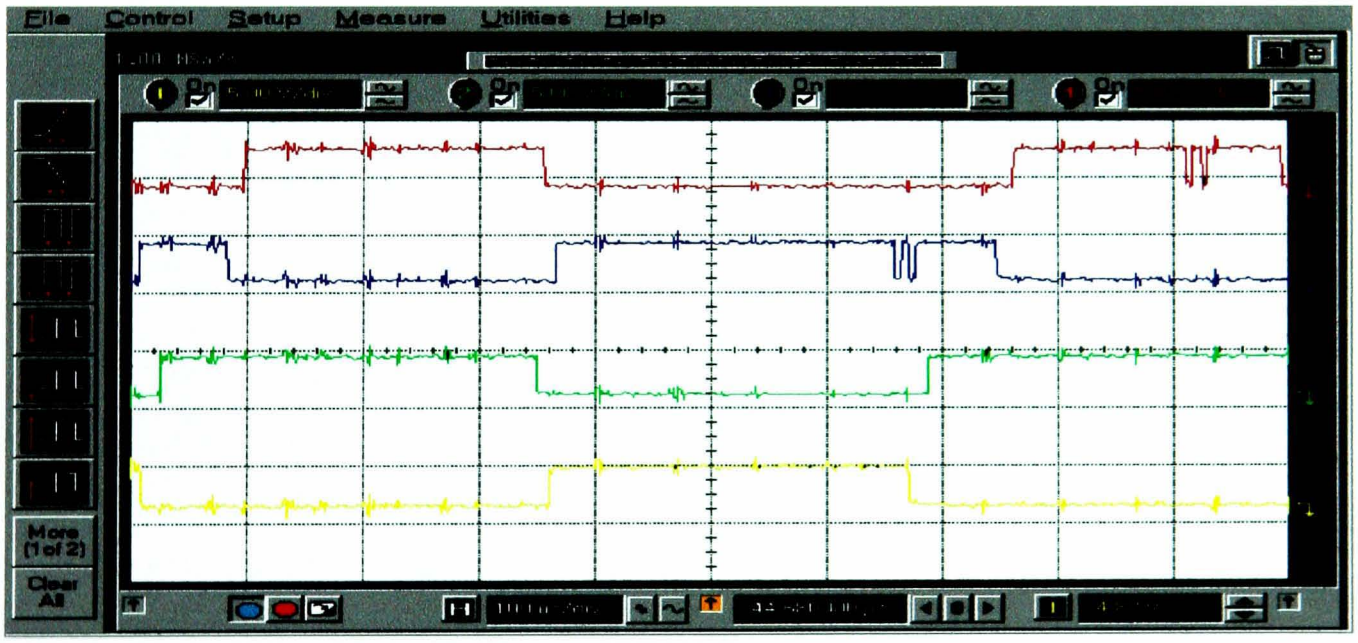


Figure 7.24 - *The switching delays produced by the FPGA to avoid the short circuits in the PWM inverter*

Employing FPGA to implement PWM strategies provides advantages such as: rapid prototyping, simple hardware and software design, higher switching frequency. The novel digital circuit realization scheme for PWM control, employing a single Xilinx FPGA, may serve either for a.c. motor drives or three-phase a.c. voltage regulation systems.

The VHDL code written to implement the PWM module for IM vector control applications allows individual mapping into an integrated circuit that can be applied to a wide range of vector control drives as well as other digital applications.

7.4 Complete System Setup and Test

The experimental set-up of the Field Oriented Controller is shown in Figure 7.25. It consists of the FH2 MKIV testbench produced by TecQuipment [**73**]. the testbench offers the facility to mount up to two electrical machines, DC and AC, on the same shaft (Figure 7.25) and includes speed and torque sensors that allow testing the motor operation. It also includes the FH90 four-pole three-phase cage induction motor. The rated line voltage is 220V while the line currents have values of up to 1A and the motor power is 0.5 KW.

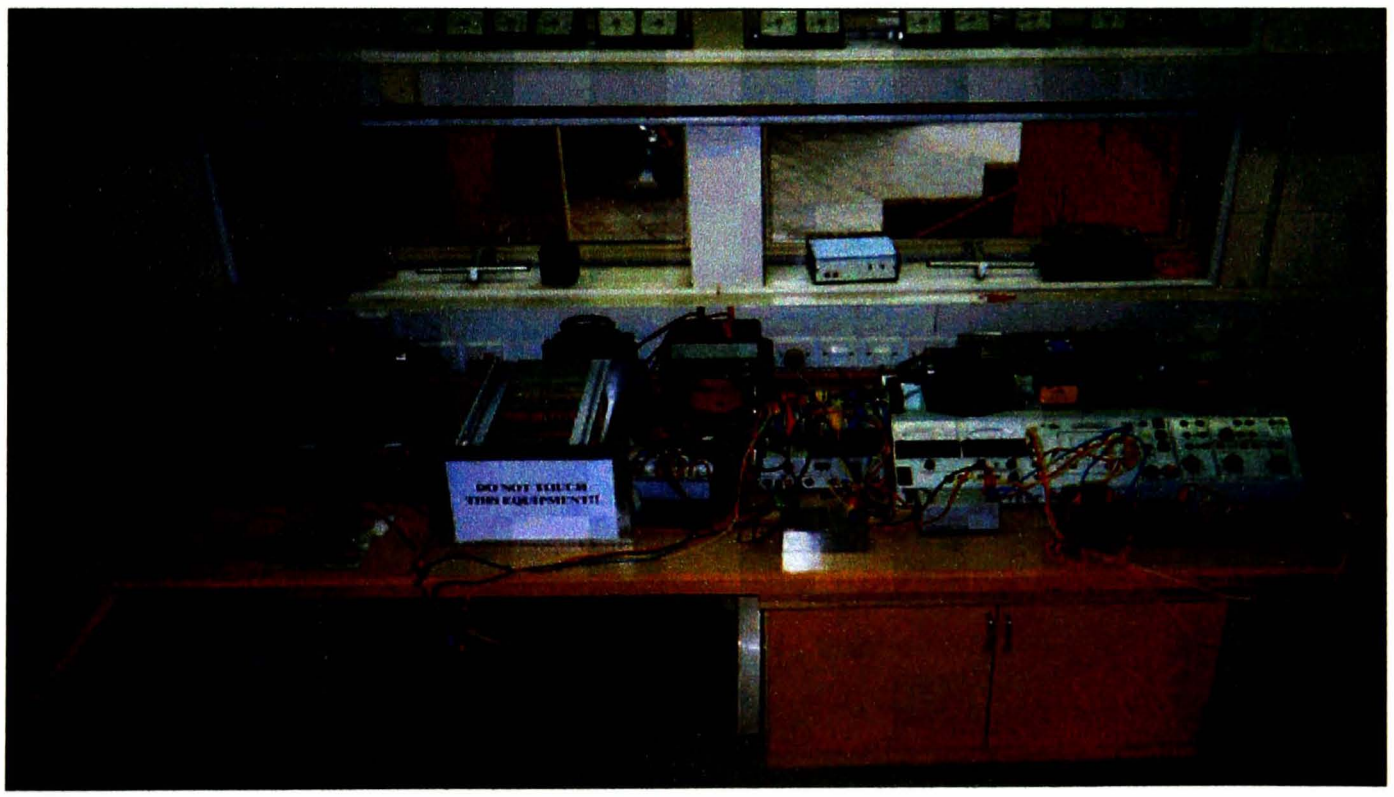


Figure 7.25 - *The FH2 MKIV testbench*

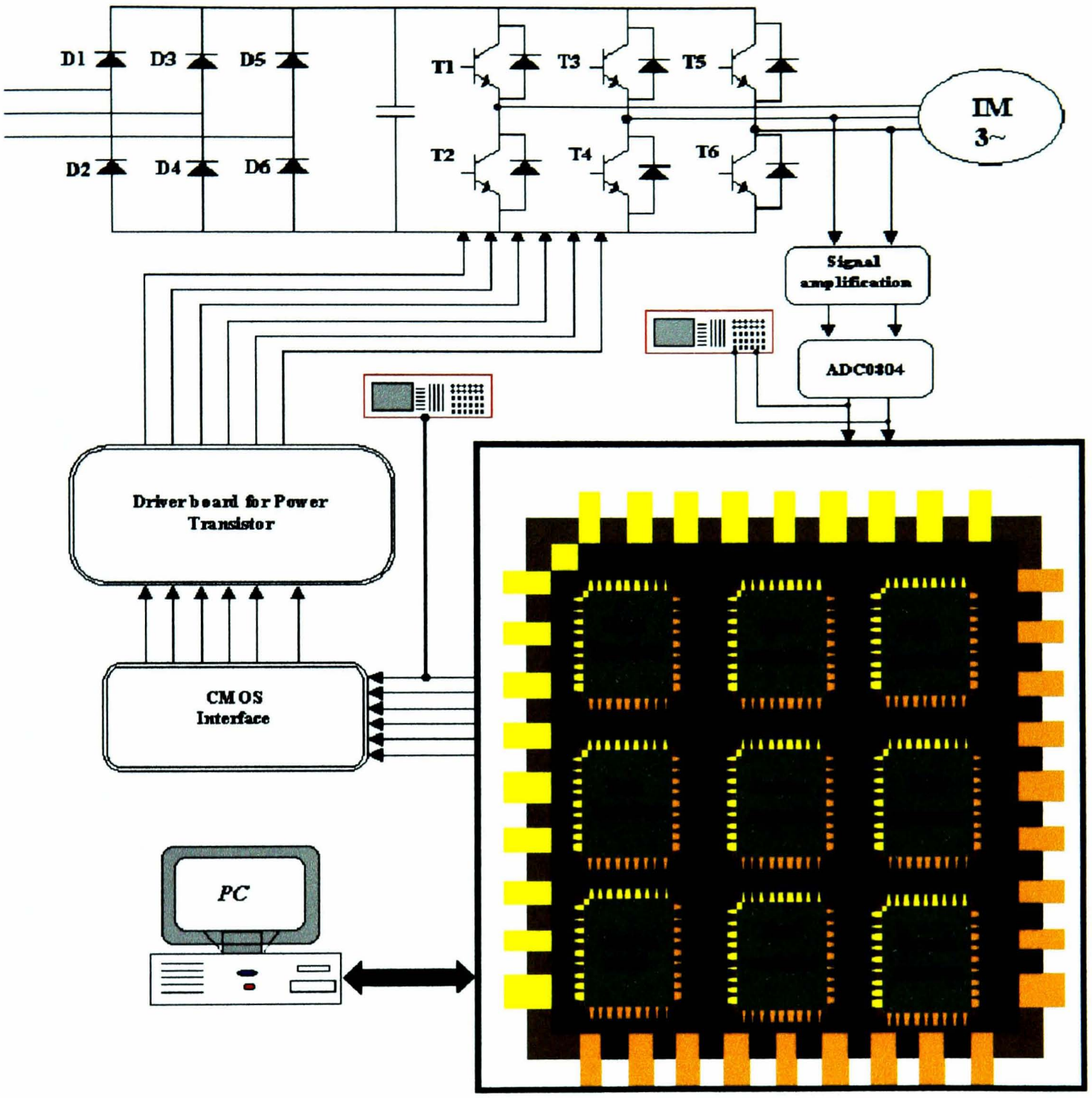


Figure 7.26 - *The Schematic of the experimental set-up*

A rectifier converts the 3 phase ac voltage into *d.c.* voltage. The *d.c.* link voltage is then inverted to sinusoidal *a.c.* output applied to the motor. There is a variety of power devices switching techniques available to invert this intermediate *d.c.* link voltage into a variable *a.c.* output voltage and frequency. Figure 7.26 shows the full wave diode rectifier bridge.

The inverter converts the *d.c.* voltage into adjustable frequency and adjustable voltage *a.c.* output. During each cycle of the fundamental frequency, the transistors are switched on and off, to generate a variable voltage output. The power devices located in the inverter section are Insulated Gate Bipolar Transistors (*IGBTs*), which have a very fast switching frequency rating. Each (*IGBT*) has a maximum collector-emitter voltage rating of 1200 *Volts* and a continuous collector current rating of *40 A* (at case temperature of 25^{o} C). The interface circuits used are illustrated in Figure 7.27 together with the Spartan S40PQ208 FPGA board Appendix (D).

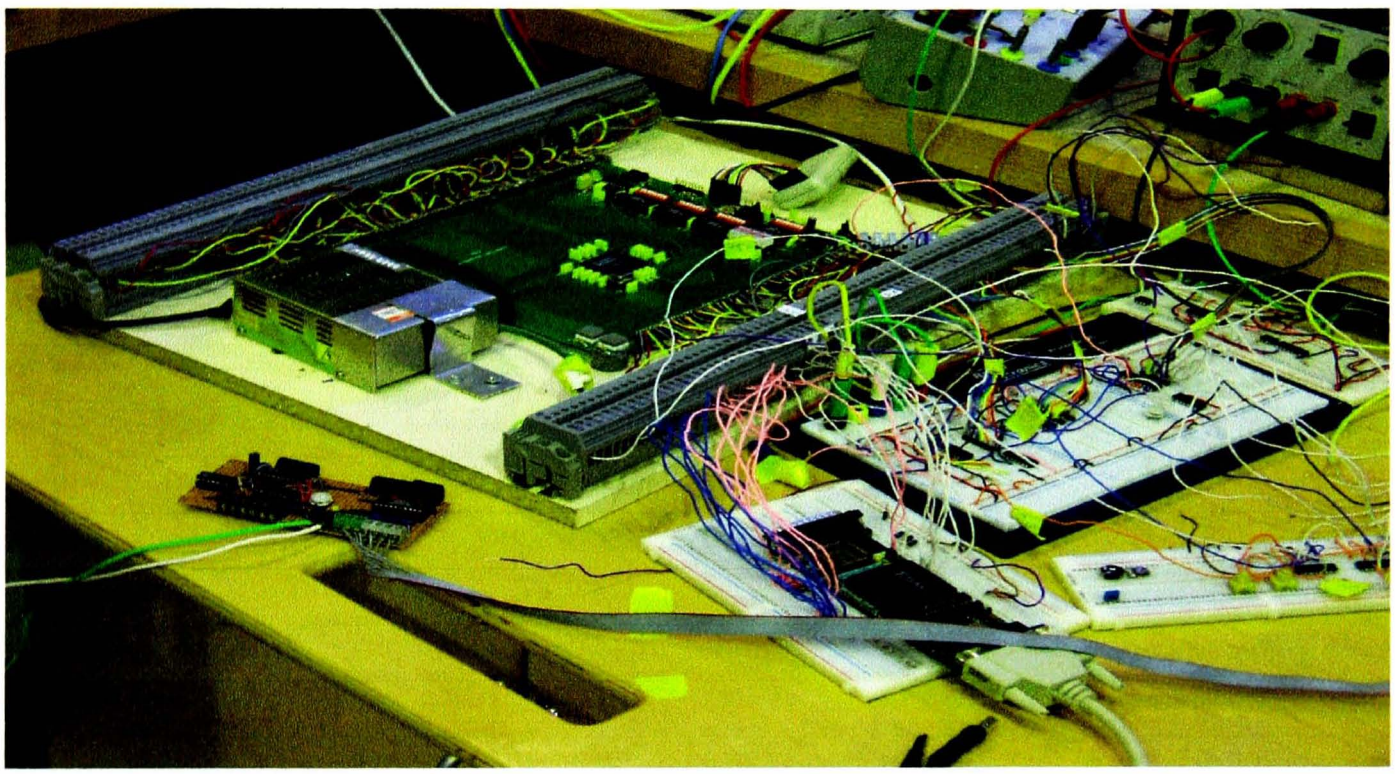


Figure 7.27 - *The Spartan S40PQ208 board and the interface board*

The *CMOS* interface circuit, illustrated as a block in Figure 7.26, amplifies the output signals of the *FPGA* to *5V* and supplies them to the transistor driver board. As these pulses have *5V* in amplitude they will go first to a boost circuit (the driver board), which brings them to the *15V* level required to drive the *IGBTs*. The conversion of the analogue signal from the transducer into digital signal will be achieved using National Semiconductor's *ADC0804 IC*. The device is an 8 bit *CMOS* Analogue to Digital Converter (*ADC*). The logic inputs meet both TTL and CMOS specifications.

It also has an on chip clock generator which eliminates the necessity for an external clock; the conversion time is 100μs. Figure 7.28 shows a simplified schematic diagram of the conversion circuit utilising the ADC0804. The motor phase currents will be measured by Hall-effect based sensors and their voltage output is sent to the *ADCs*. However, as the Hall sensors output may be considered in the range *–2.5* to *2.5*V and the *ADC* can only digitise voltages from 0 to *5*V, an auxiliary circuit (Figure 7.29) was built.

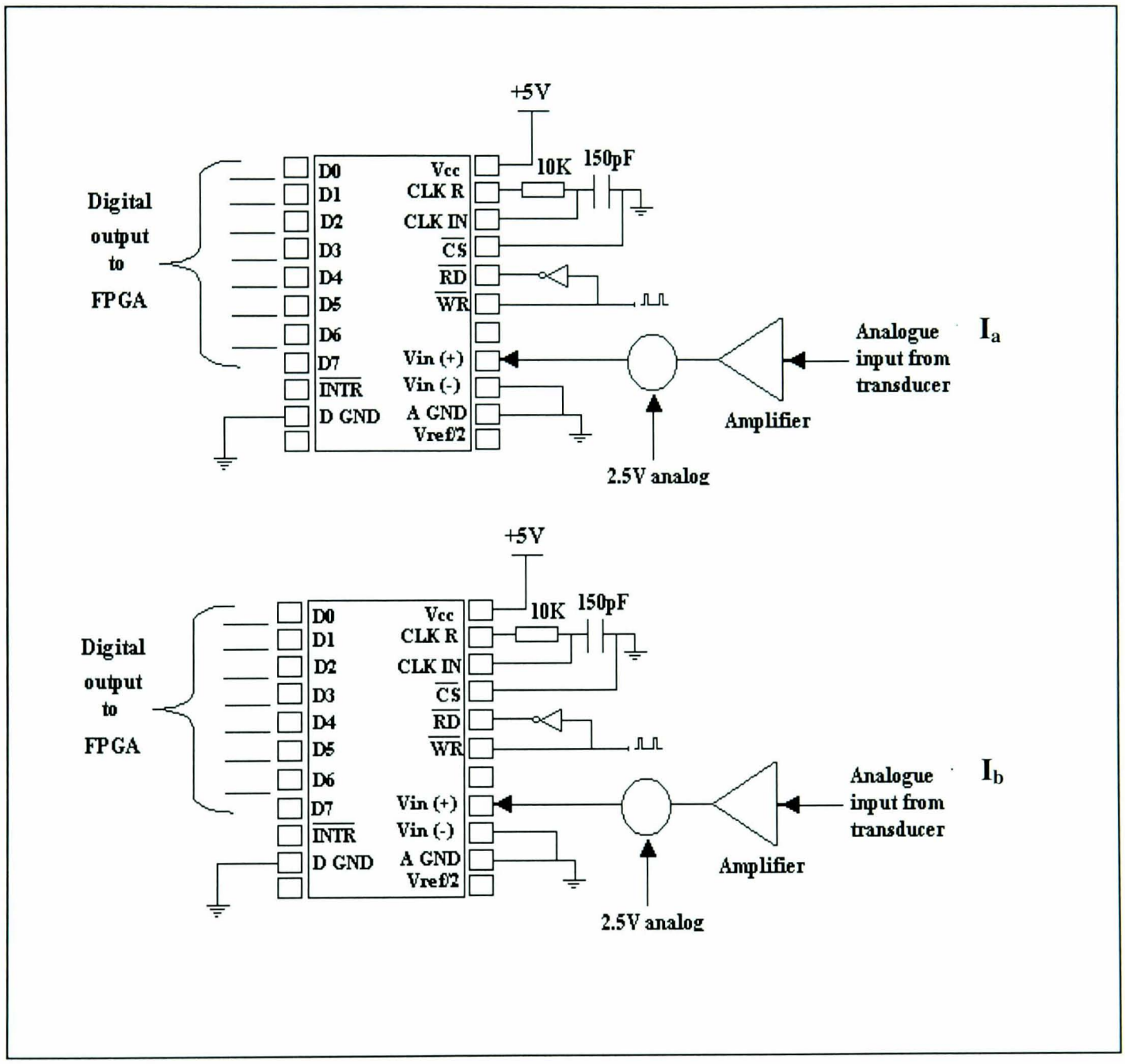


Figure 7.28 - *Analogue to digital converter*

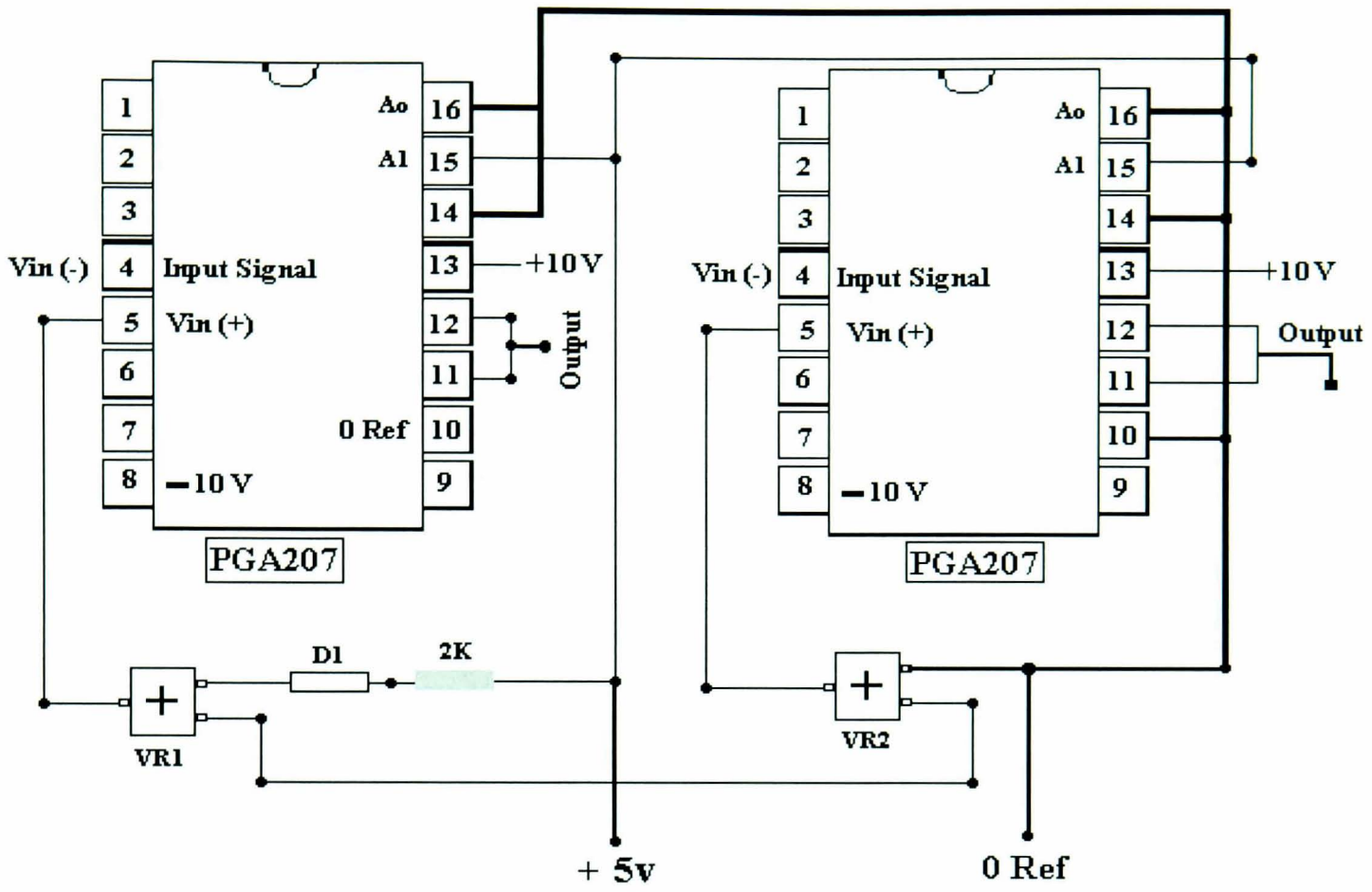


Figure 7.29 - *Voltage shift circuit*

The circuit is based on PGA207 IC. The device is a digitally programmable gain instrumentation amplifier, suitable for data acquisition systems [74]**.** This circuit realises the offset of the Hall sensors voltage output to a range of 0 to 5V.

The digital inputs A_0 and A_1 select the gain according to the logic gate in table 7.5.

PGA207	A_1	A_0
1	0	0
2	0	1
5	1	0
10	1	1

Table 7.5 - *Gain selection*

The signal from the Hall-effect based sensor is connected to pin (4) of the PGA207 while the output signal of the PGA207 pin (11) is connected to the Vin(+) pin of the A/D converter. The digital output is fed into the FPGA via a buffer. The signals connected to RD and WR pins (Figure 7.28), come from a clock and pulse generation circuit (Figure 7.33). Figure 7.30 shows the actual measured phase current of the motor and Figure 7.31 illustrates the same signals amplified by a factor of 2. The experimental results shown in Figure 7.32 clearly indicate the signal applied to RD and WR.

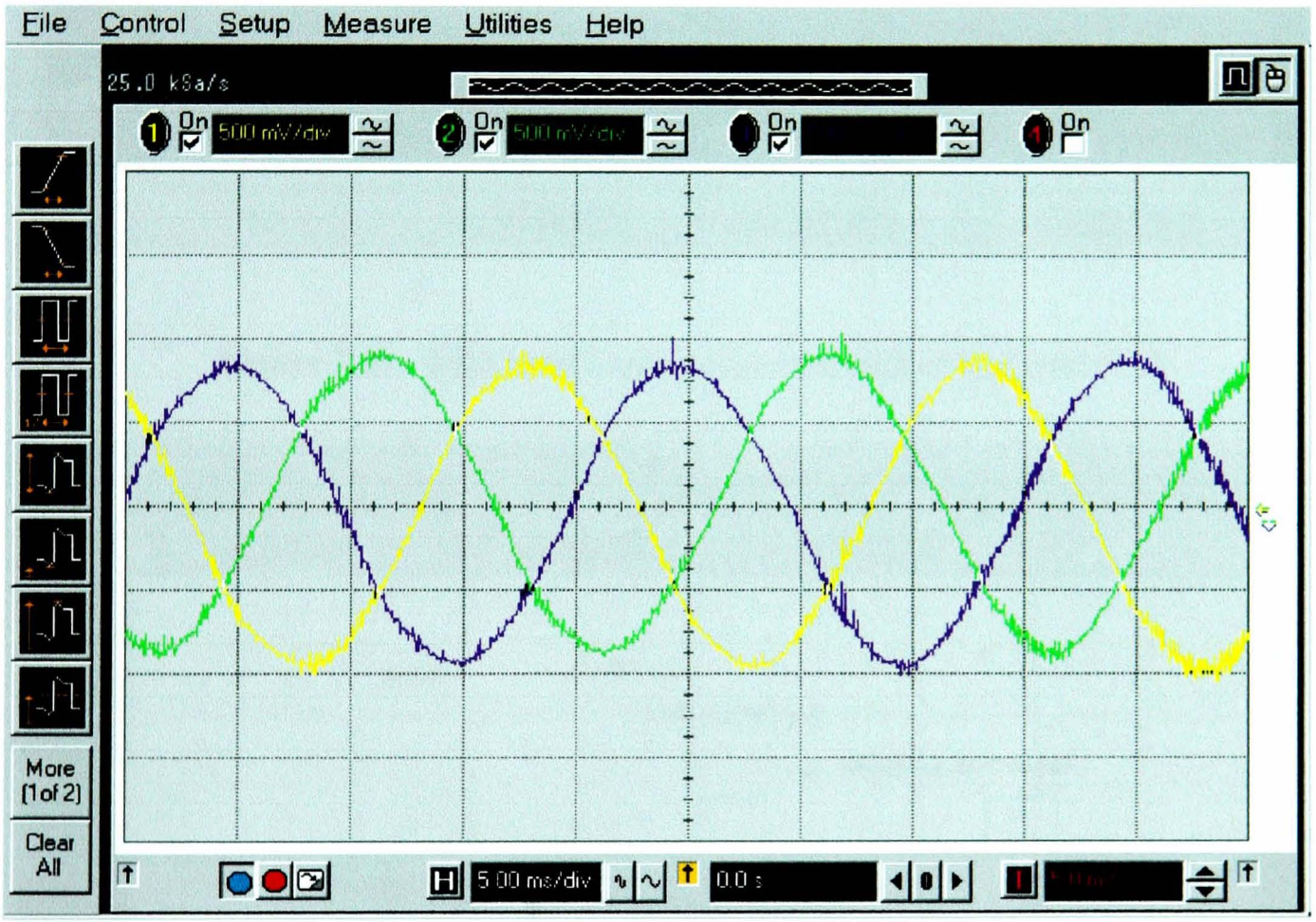


Figure 7.30 - *Measured i_a, i_b and i_c currents by the Hall sensors*

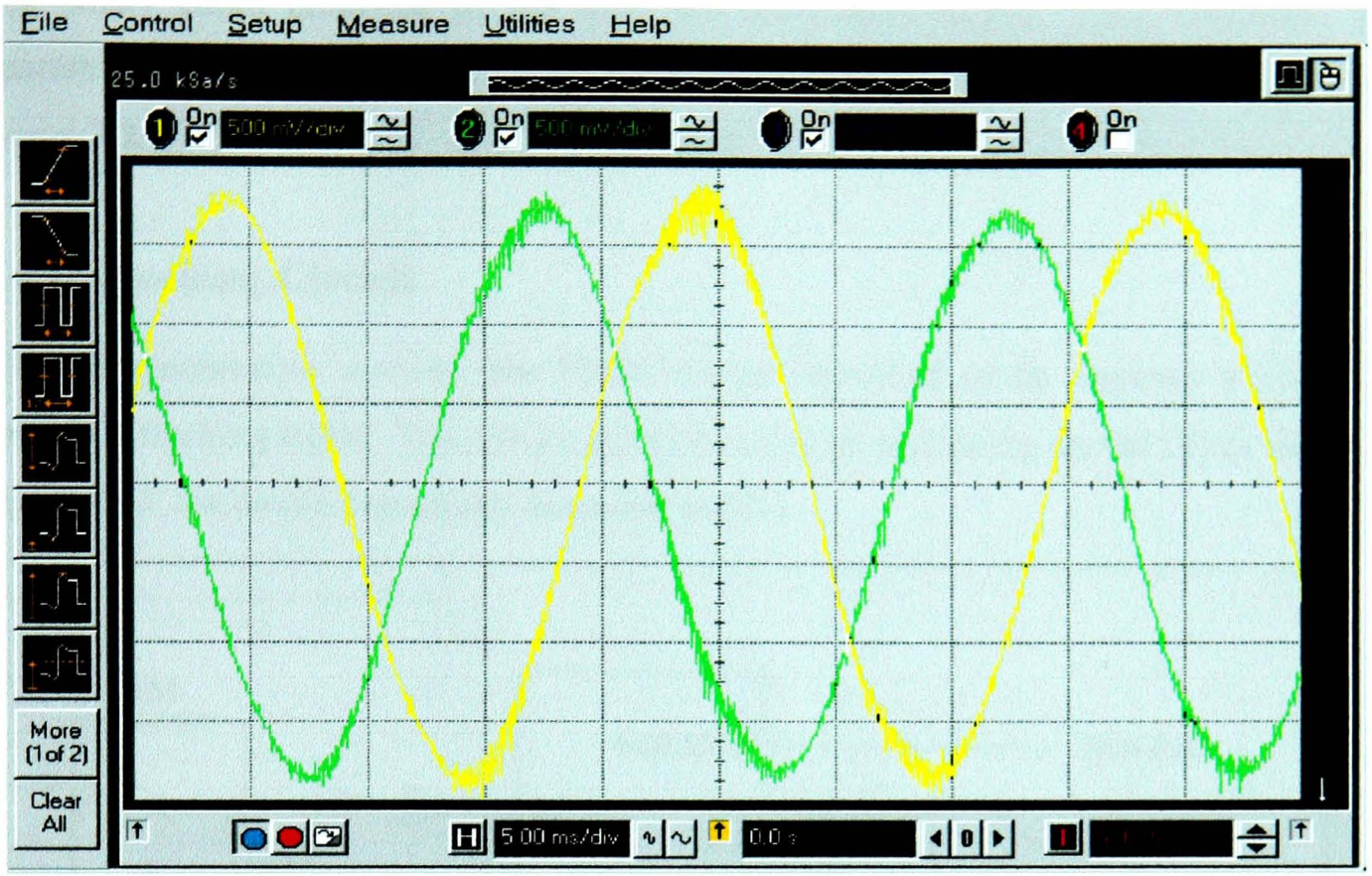


Figure 7.31 - *Measured i_a and i_b currents shifted by factor of 2*

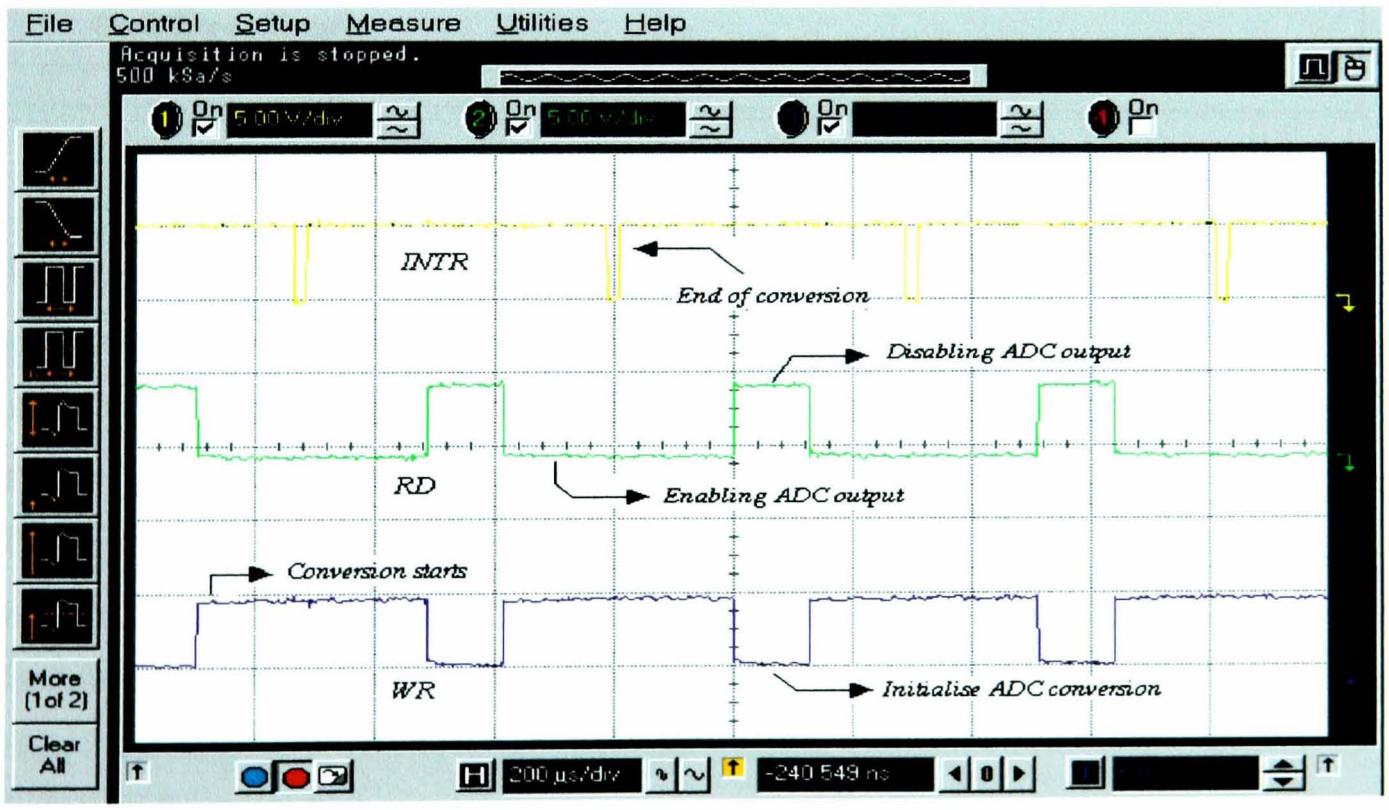


Figure 7.32 - *The binary codes produced by the ADC*

The binary codes produced by the *ADC* are transmitted to the *FPGA* controller. The Spartan S40PQ208 FPGA design is partitioned into natural abstract blocks, as illustrated in Figure 7.2.

7.4.1 Clocking Circuit

Like all synchronous circuits, the FPGA design described so far requires a circuit to provide a clocking signal. This can be achieved using an oscillating crystal (*X*) as shown in Figure 7.33, the circuit being fully described in [75].

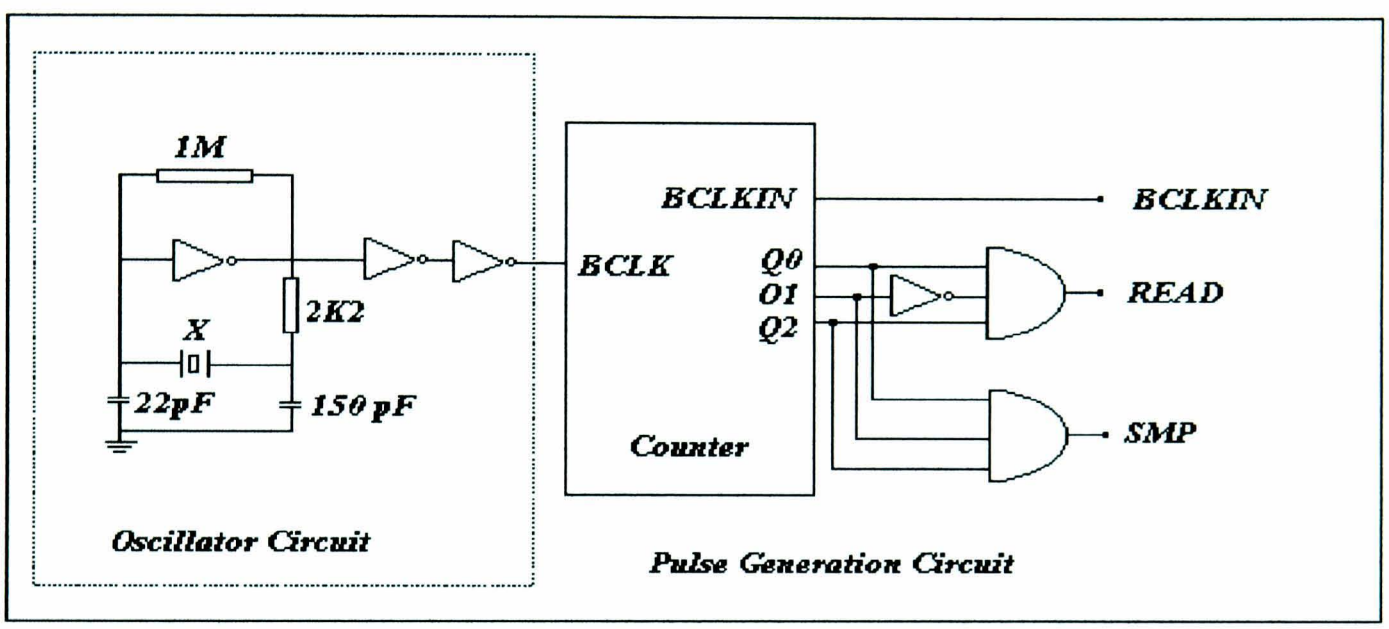


Figure 7.33 - *Clocking circuit*

The circuit in Figure 7.33 is divided into two sections, an oscillator circuit and a pulse generation circuit. The former is a standard CMOS clock signal generation circuit, which uses a crystal to provide the oscillation, while the latter generates the appropriate clock and pulse sequence for the FPGA and A/D converter.

The entire pulse generation circuit can be accommodated into the FPGA design. In order to achieve this, a VHDL model of a clocking circuit in Figure 7.33 was created. As shown by the following VHDL code, the clocking circuit is an entity having one input *BCLK* and the three main outputs *BCLKIN, READ* and *SMP*.

```
ENTITY Bclk_circuit IS
 PORT(Bclk:in std_logic;
     BCLKIN,cc:inout std_logic;
         READ,Q0,Q1,Q2,Q3:out std_logic;
         SMP:out std_logic);
```

```
END Bclk_circuit;
ARCHITECTURE Bclk_circuit_arch OF Bclk_circuit IS
      signal counter: integer range 0 to 1000;
      signal count: integer range 0 to 500;
      signal co: STD_LOGIC_VECTOR(3 DOWNTO 0);
      signal a: STD_LOGIC;
begin
process (Bclk,counter)
 begin
   if Bclk='1' and Bclk'event then
     counter <=counter+1;
     count<=count+1;
   end if;
   if counter <500 and counter >=0 then
     BCLKIN<='1';
    elsif counter <1000 and counter >=500 then
     BCLKIN <='0';
   end if;
   if count<250 and count>=0 then
     cc<='1';
     elsif count>=250 and count <500 then
     cc<='0';
   end if;
   if counter >1000 then
    counter<=0 ;
    count<=0;
   end if;
end process;
process (cc,co)
 begin
  if cc='1' and cc'event then
     co <=co+'1';
     Q0<=co(0);
     Q1<=co(1);
     Q2<=co(2);
     Q3<=co(3);
     READ<= co(0) and not co(1) and co(2);
     SMP<= co(0) and  co(1) and co(2);
```

```
 end if;
end process;
END Bclk_circuit_arch;
```

The signal *BCLK* is used as the primary clocking signal for the FPGA. The *READ* pulse initiates the A/D conversion process in the ADC0804, while the *SMP* pulse enables the FPGA to sample the converted digital signal from the A/D converter. The sequence of these two pulses is such that the FPGA takes a sample of the digital value at the output of the converter just after conversion. Based on the conversion time of the ADC0804, it is necessary for the time lapse between the *READ* pulse and the *SMP* pulse to be more than 100μs for the conversion to be complete. This corresponds to a maximum clock frequency of 10KHz. A waveform diagram of the pulse generation circuit outputs (*BCLKIN, READ* and *SMP*) is shown in Figure 7.34. Figure 7.35 shows the status report of the implementation process.

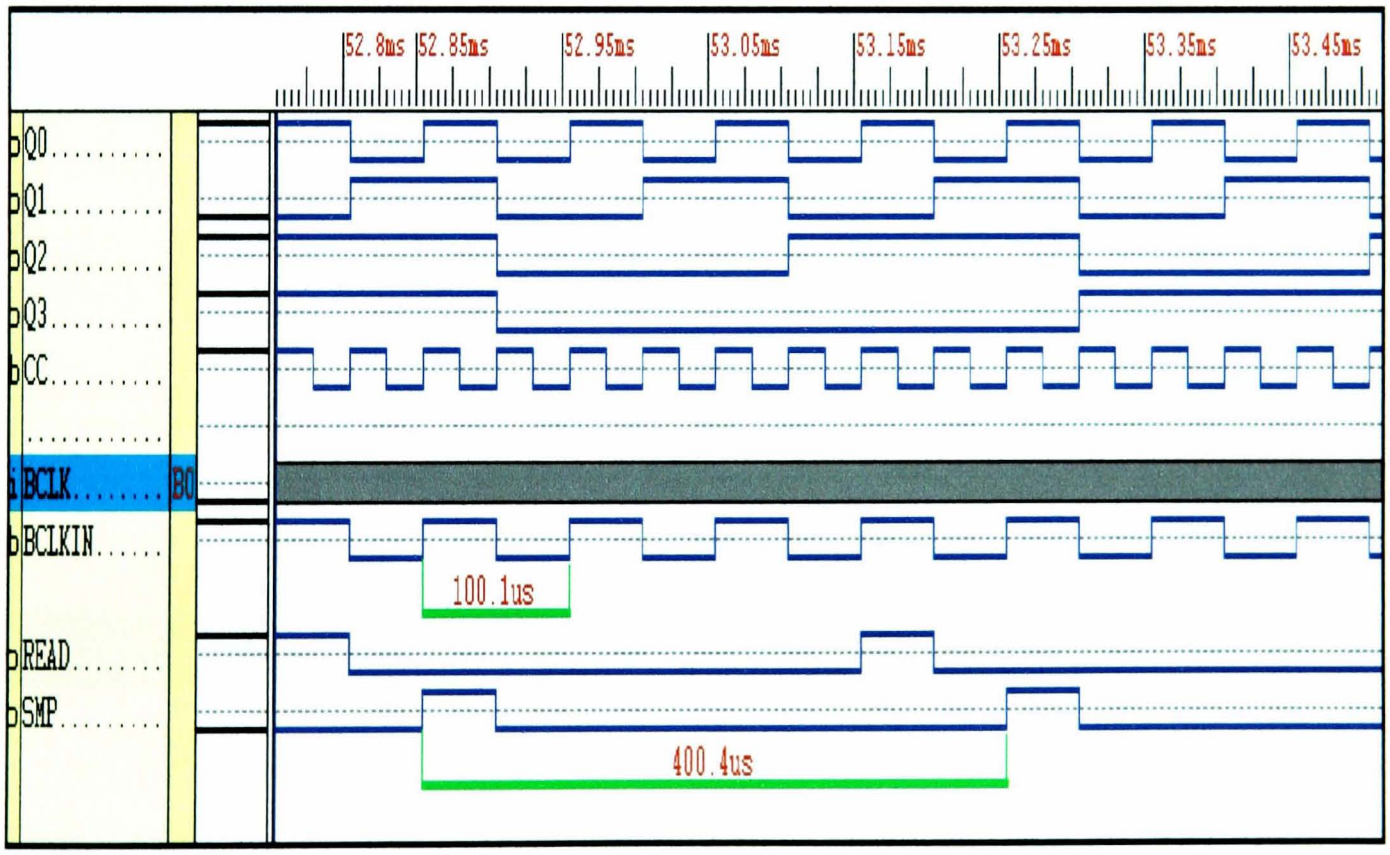


Figure 7.34 - *Waveforms of pulse generation circuit.*

```
Design Summary
--------------
   Number of errors:          0
   Number of CLBs:                19 out of    784     2%
      CLB Flip Flops:         23
      CLB Latches:             2
      4 input LUTs:           34
      3 input LUTs:            2
   Number of bonded IOBs:         9 out of    169     5%
      IOB Flops:               6
      IOB Latches:             0
   Number of clock IOB pads:      1 out of      8    12%
   Number of BUFGLSs:             4 out of      8    50%
Total equivalent gate count for design: 516
```

Figure 7.35 - *Implementation design summary for Clocking circuit*

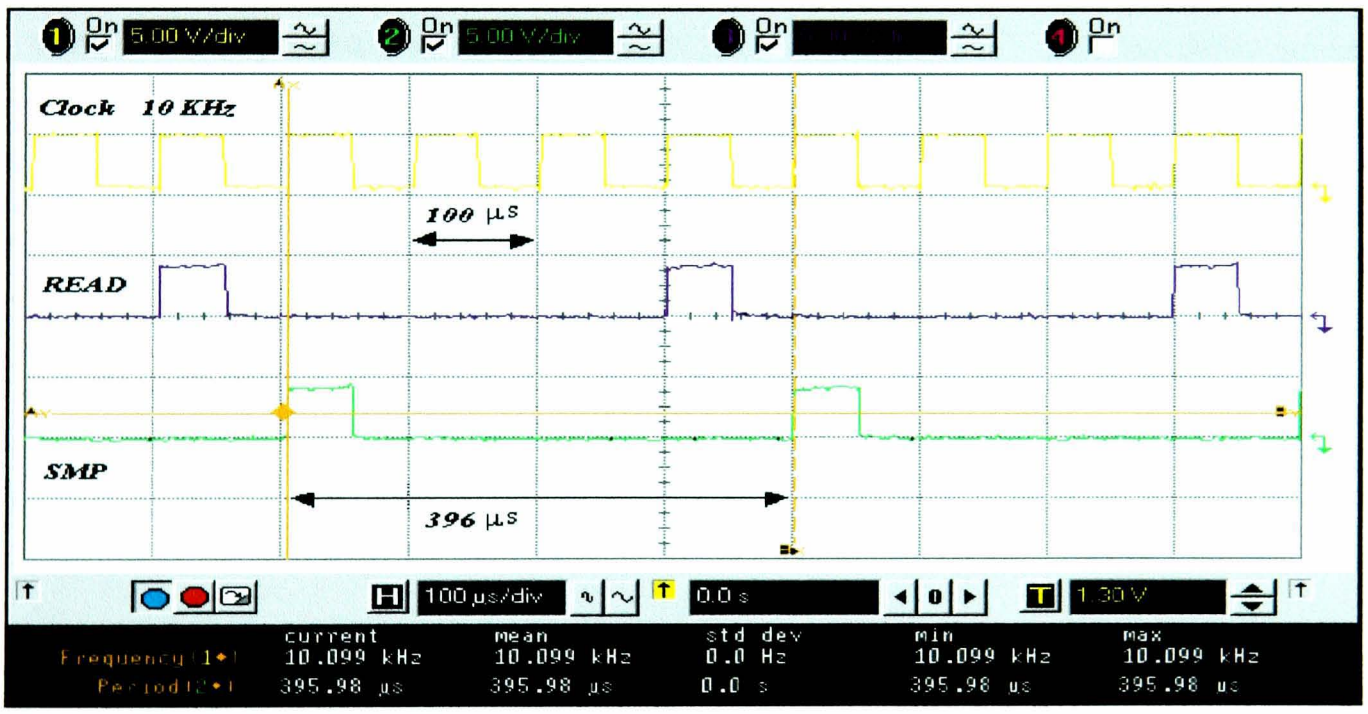


Figure 7.36 - *Implemented waveform of pulse generation circuit*

Figure 7.36 illustrates the results of the downloading of the pulse generation component design in the Xilinx FPGA. It can be seen that the practical results are identical to the simulated values shown in Figure 7.34.

7.4.2 Overall experiment test

The overall system experimental testing is shown in Figure 7.37 and is based on measuring the motor phase currents using the Hall-effect based sensors. Their voltage output is sent to the A/D converter on the interface board. However as the Hall sensors output may be considered in the range –1 to +1 and the A/D converter can only digitise voltages from 0 to 5V, an auxiliary circuit was built. Consequently, motor currents are measured and are taken as external input data for the control scheme. In addition, a speed estimation encoder is built inside the induction motor. As a result of the control algorithm being based on current and speed measurements, an external speed reference is included in the VHDL code as a constant for experimental purposes. Figure 7.38 presents four of the PWM control signals generated by the FPGA motor controller. They have been monitored using a four-channel Hewlett Packard digital oscilloscope. The figures demonstrate the correlation between two of the signals controlling the transistors on the same inverter leg. Thus, the two signals have complementary values: when one of them is '0' the other is '1'. The 5μs delay generated by the interface block is not visible in Figure 7.39 due to the inappropriate time scale (500μs/div), but it can be observed in Figure 7.40 and Figure 7.41, where the time scale is (10μs/div). The control scheme generates six PWM outputs. These pulses, of 5V amplitude, will go first to a boost circuit, which brings them to a 15V level, thus permitting them to drive the IGBTs.

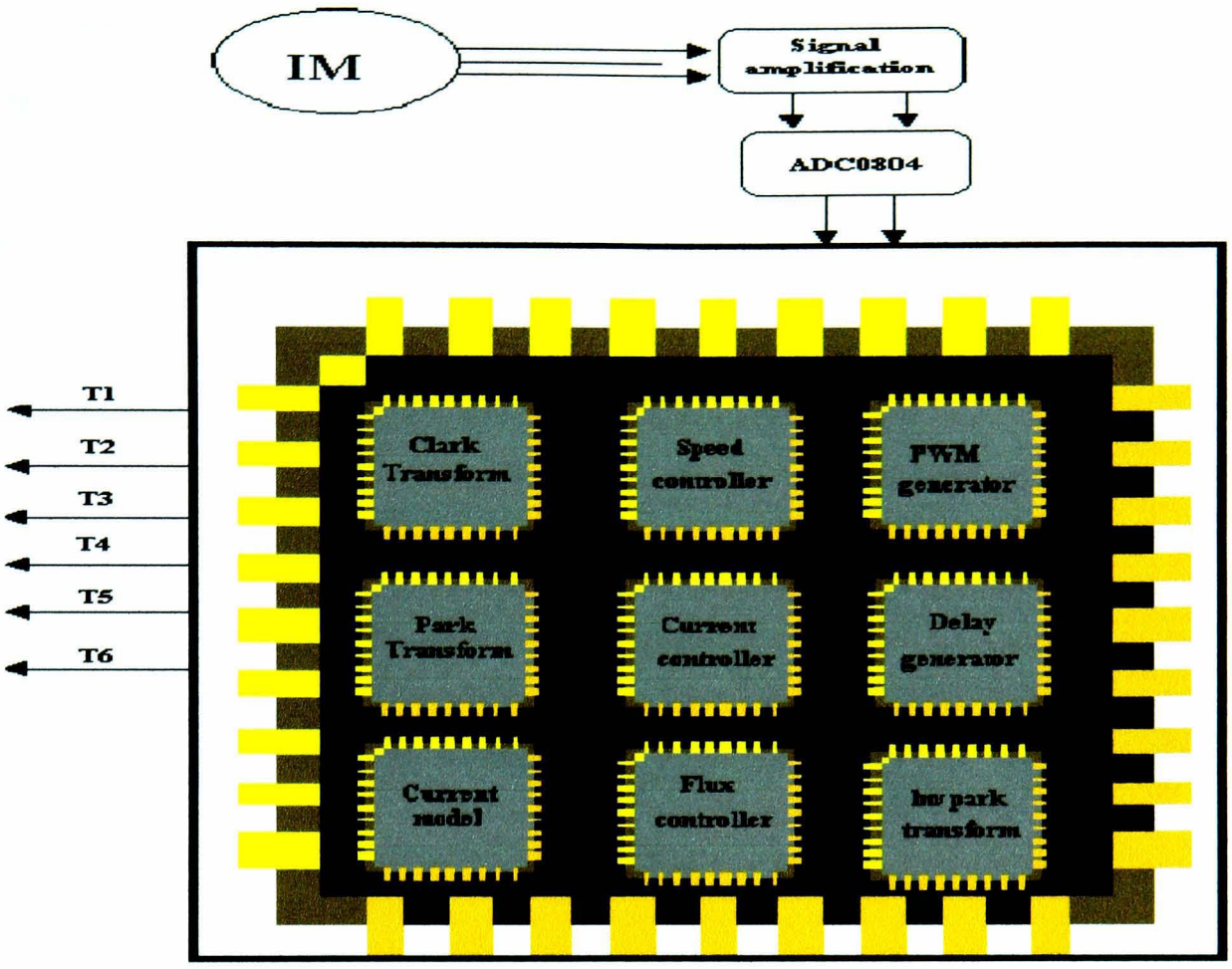


Figure 7.37 - *The Overall experiment*

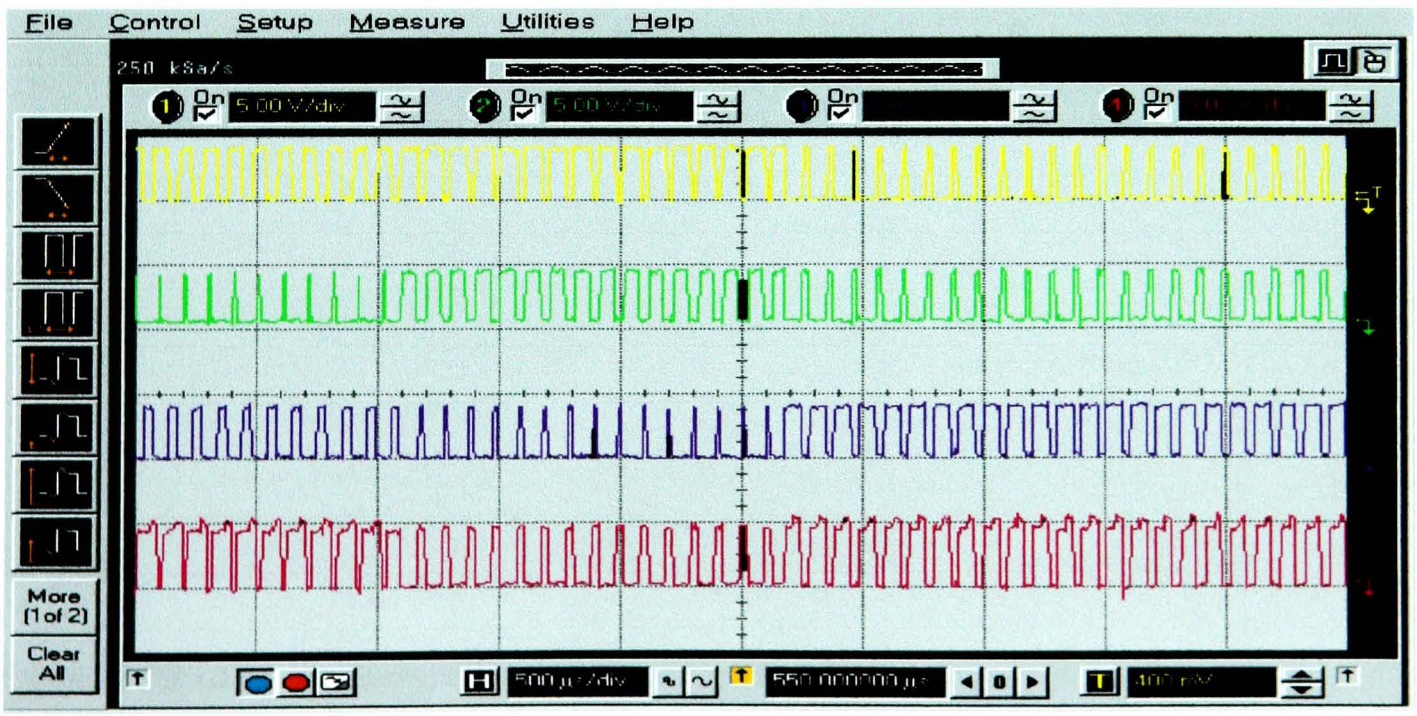


Figure 7.38 - *PWM outputs signals*

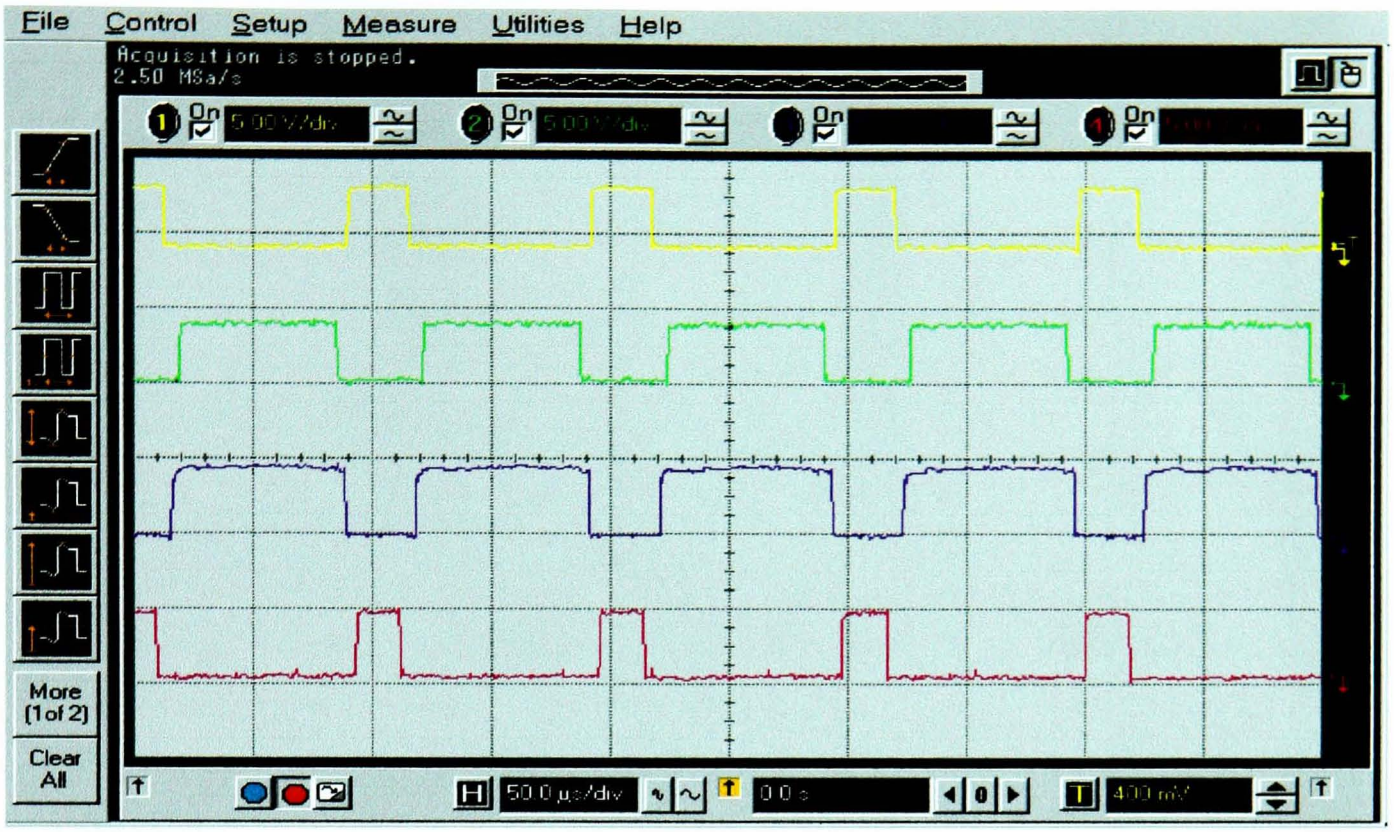


Figure 7.39 - *Four of the FPGA output signals controlling the IGBTs*

Figure 7.40 - *The switching delay produced by the FPGA to avoid the short circuit for T1 and T2*

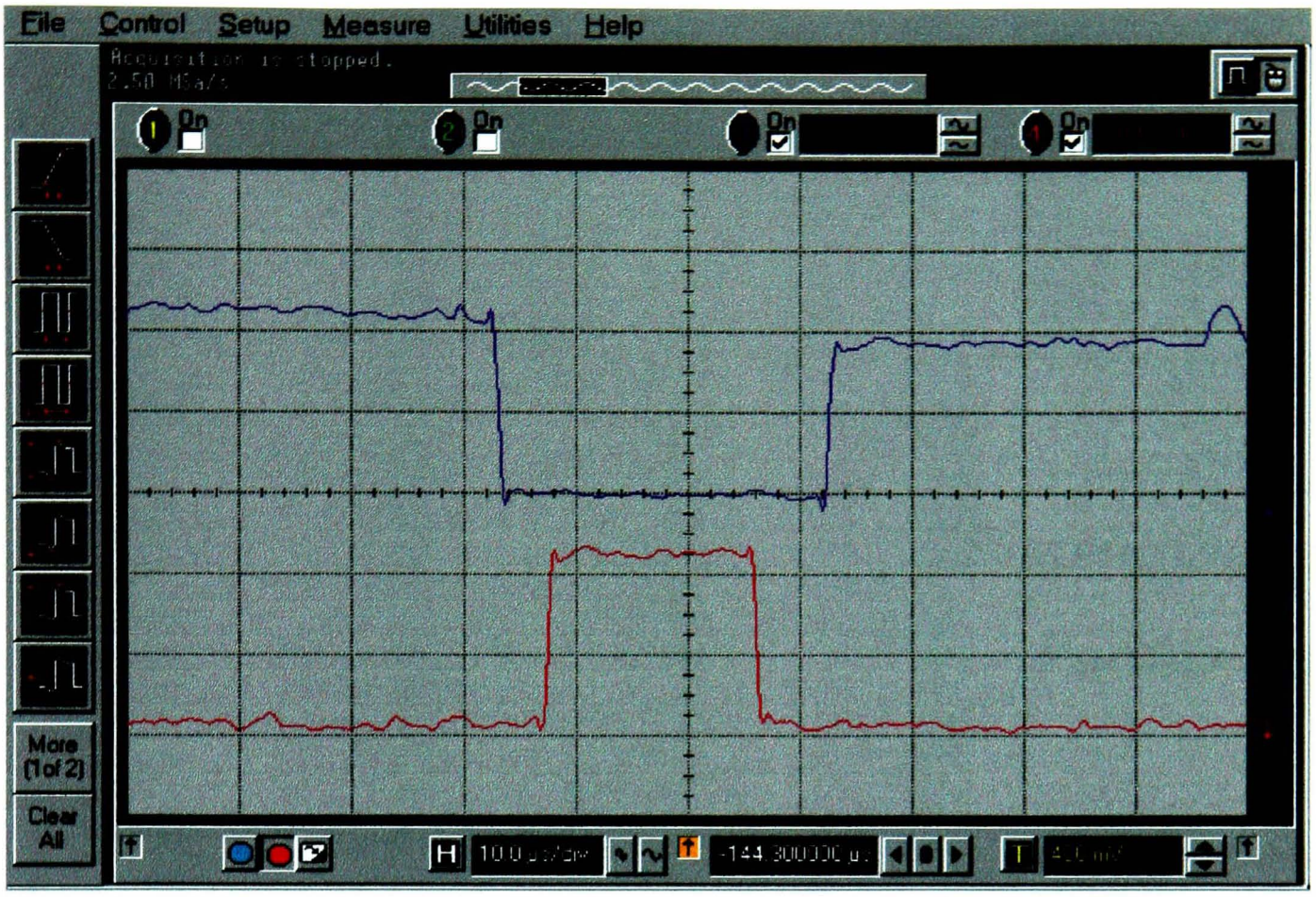


Figure 7.41 - *The switching delay produced by the FPGA for T3 and T4*

The test results confirm that the designed controller is operating correctly on conjunction with the interface circuits and the power system. The new modelling technique, based on hierarchical design and modular development, is therefore validated by both simulation results and the experimental tests carried out after implementations.

A novel design and development methodology of vector controlled power systems using VHDL was established. Next chapter presents some conclusions on the work as well as further developments to the system that are being investigated.

Chapter 8

Conclusions and Further Work

A new approach to the modelling, simulation, digital controller design and implementation for induction motor electric drive systems was successfully developed. The approach uses Very high speed integrated circuit Hardware Description Language (VHDL) as a unique EDA environment for system modelling, evaluation and controller design. Simulation and experimental results are presented, proving the validity of the novel approach when applied to a vector controlled induction motor system. The advantages of using VHDL/FPGA for drive modelling and control strategy implementation are enumerated.

8.1 CONCLUSIONS

General research in the area of induction motor control has identified a large number of control strategies, based on different control observation and adaptive algorithms combined with different coordinate systems, state variables and notations. From the control point of view, induction motors are non-linear high order systems of considerable complexity. Due to the high mathematical content of the modeling equations, the signal processing performed by the control systems is complicated and normally requires expensive hardware.

Although most a.c. drives in use today adopt a microprocessor based digital control solution, the implementation of the current loop and PWM control is still tied to analog control circuitry.

This kind of control scheme has the advantage of fast dynamic response but suffers the disadvantages of circuit complexity, limited functions and difficulties, when circuit modification is required.

DSP technology has generally been used for digital a.c. motor control implementation, however, recent developments in the field of microelectronics, have enabled complex switching strategies for transistorized inverters to be implemented by means of ASIC technology. Consequently, motor control and power conversion systems employing ASICs / FPGAs are receiving increased attention.

In order to achieve the performance required by servo applications, induction motors are controlled using vector control strategies. The new methodology presents a comprehensive study of the integrated modelling, simulation and design of a vector controlled induction motor using VHDL and targeting FPGA implementation. Although VHDL is a hardware description language and as such is primarily used for circuit design, it has the basic properties of any software programming language. This allows complete power system modelling and simulation, prior to finalizing the digital controller design and implementation. The top-down methodology consists of hierarchically modelling the system at all levels of abstraction, ranging from the behavioral level to the logic gate level. The drive system model used enables the high performance control of speed and torque.

When designing and modelling digital systems in VHDL, the design is partitioned into natural abstract blocks, known as components. The complete system is then modelled using a hierarchy of components, known as "design hierarchy". Each element of the Vector Control Modules is designed and carefully optimised for synthesis. Seven VHDL components were identified and designed: Clark_transform, Park_transform, Current_model, PI_controller, Control_unit, Inverse Park_transform and PWM generation. Their representation in VHDL has been described, followed by FPGA implementation. The PWM control features and performance characteristics are then demonstrated and the overall strategy is validated by experimental results. The VHDL digital controller design

was finally downloaded to a single Xilinx Spartan XCS40PQ208 FPGA, containing the equivalent of 40000 logic gates.

The major advantages of the new approach, are:

- It provides a unique environment for modelling, simulation and evaluation of complete drive systems, including controllers, power electronics and induction motors.
- the same environment (VHDL) is used for the design of the digital controller achieving the vector control strategy and for silicon (FPGA) implementation.
- fast design development and short time to market.
- a CAD platform independent model and design are developed (VHDL operates with ASCII files) and therefore valuable IP can be produced, in co-relation with the modern principles of design reuse.
- That concurrent engineering basic rules (unique EDA environment and common design database) are fulfilled.
- compact and fast FPGA solution for implementation.
- high accuracy and speed of response of the controller.

In addition, the universal applicability (reusability) of these control blocks to a wide range of vector control drives is an important asset. The complete digital controller circuit design has been synthesized, implemented and tested. The test results confirm that the designed controller operates correctly in conjunction with the interface circuits, the power system and the induction motor. The new modelling technique, based on hierarchical design and modular development, is therefore validated by both simulation results and the experimental tests carried out after implementation.

The extensive use of VHDL for drive systems modelling and simulation, in conjunction with accurate digital controllers design is foreseen as a practicality in the near future. This work has pioneered the approach.

8.2 ORIGINAL CONTRIBUTION TO THE THESIS

The original achievements of the present research work can be summarized as:

- The development of a new approach to the modelling, simulation and controller design of a complete induction motor electric drive system. The novel technique uses a hardware description language (VHDL) as a unique EDA environment for the system modelling, evaluation and controller design.
- The design and simulation of an original vector controller using VHDL for controlling the speed of the induction motor.
- The vector controller was downloaded into Xilinx FPGA Spartan (XCS40PQ208) for rapid prototyping before being comprehensively tested.
- The production of a set of reusable VHDL models, in accordance with the modern principles of design reuse, which can be incorporated in a range of novel vector controllers. These are:
    - Clark transform
    - Park transform
    - Current model
    - PID Controller model
    - Inverse Park transforms
    - PWM Generator using VHDL.
    - Multiplier
    - Divider
- The design of a range of extra digital and analogue circuits as interfaces for control system commissioning.

From the outcomes of the research, it can be concluded that all the original objectives are met.

8.3 Further Work

This section discusses possible improvements to the system and areas which warrant further investigation. Two developments of the system have been considered: the embedded system controller implementation and the sliding mode control technique.

8.3.1 Embedded System

Based on the end user's application needs, FPGAs and DSPs can be utilised separately or combined, in order to form powerful high performance digital control capabilities, as shown in Figure 8.1.

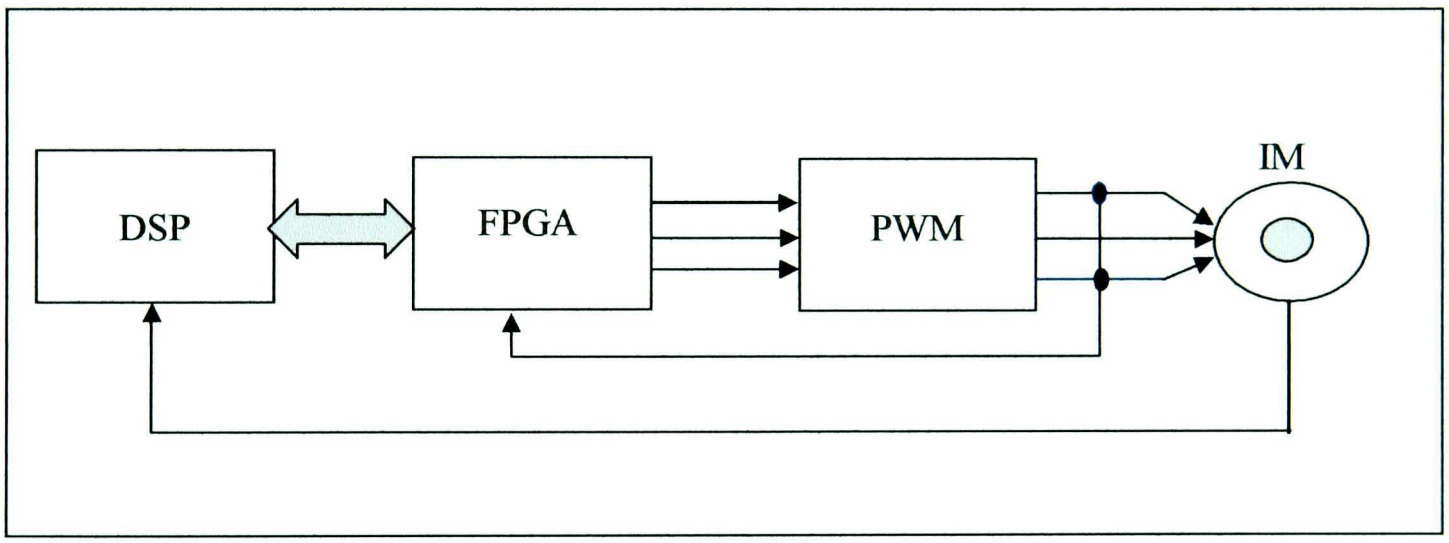


Figure 8.1 - *DSP/FPGA based digital control*

The embedded implementation of the digital control system could provide important advantages by combining the strong features of each of the two devices. These are:

DSP Strengths:

- Ideal for extremely complicated math-intensive tasks.
- Ideal for numerous conditional operations.
- Best at low data rates: 10's Mbytes/sec or less.
- Suitable for fixed or floating-point computations.

FPGA Features:

- Ideal for rigid and repetitive tasks.
- Not so good for nested conditional operations: if, then, else, etc…
- Ideal for application with algorithmic logic level parallelism (e.g. multiple ALUs).
- Ideal for higher data rates 100's Mbytes/sec.
- With gate densities as high as 1 million gates, a variety of complex operations historically within the domain of DSP processors, can now be implemented on FPGAs.

8.3.2 Sliding Mode Control

Another possible development considered is the use of the sliding mode control technique, which could employ an embedded system for implementation. The embedded system could be divided into two parts: the first implemented in FPGA that includes any fixed module of the controller requiring parallel processing at high speeds, the second, containing the elements of design that require frequent changes, implemented in DSP.

Current research in the control of induction motors is characterized by a great variety of control methodologies with different control, observation and adaptation algorithms combined with different coordinate systems, different state variables and different notations. For the sliding mode control design to be investigated, the dynamic model given in the orthogonal stator coordinate frame (α,β) is mainly used, with stator current components and rotor flux components as state variable, complemented by the mechanical equation. As of importance is the speed, stability, dynamic response and cost of control solutions [76].

In order to apply sliding mode control, let $\omega_r^*(t)$ be the reference rotor speed and $e = \omega^* - \omega$ be the speed tracking error. If state variables $x_1 = e$ and $x_2 = \dot{e}$ are defined, then the motion equation can be represented, with respect to the x_1, x_2 state, by:

$$\dot{x}_1 = x_2 \qquad \textbf{8.1}$$

$$\dot{x}_2 = -a_1 x_1 - a_2 x_2 + f(t) - bu \qquad \textbf{8.2}$$

where a_1, a_2 and b are constant values. Since equation 8.2 describes a second order system, the switching function is designed as:

$$s = cx_1 + \dot{x}_1 \qquad \textbf{8.3}$$

with c being a positive constant. The associated controller is defined as

$$u = u_0 sign(s) \qquad \textbf{8.4}$$

where u_0 is the link voltage. According to these equations, the speed tracking error x_1 decays exponentially after sliding mode occurs at $s = 0$:

$$s = cx_1 + \dot{x} = 0 \qquad \textbf{8.5}$$

where constant c determines the convergence rate. The system's motion in sliding mode is independent of the a_1, a_2, b parameters and the disturbances in $f(t)$.

In order to apply the sliding mode control to a three phase induction motor, let $\omega_r^*(t)$ be the reference rotor speed and λ_0 the reference value of the rotor flux magnitude. The two switching functions can then be defined as follows:

$$s_\lambda = c_1(\lambda_0 - \| \lambda(t) \|) + \frac{d}{dt}(\lambda_0 - \| \lambda(t) \|) \qquad \textbf{8.6}$$

$$s_\omega = c_2(\omega_r^* - \omega_r(t)) + \frac{d}{dt}(\omega_r^* - \omega_r(t)) \qquad \textbf{8.7}$$

where $\| \lambda(t) \| = \sqrt{\lambda_\alpha^2 + \lambda_\beta^2}$ is the magnitude of the rotor flux and c_1, c_2 are positive constants determining the motion performance in sliding mode.

For the current control design, the switching function is defined as the difference between the desired and the real currents. The switching functions for both current components i_d and i_q are selected as:

$$s_d = i_d^* - i_d \qquad \textbf{8.8}$$

$$s_q = i_q^* - i_q \qquad \textbf{8.9}$$

where i_d^* and i_q^* denote the desired values for currents i_d and i_q, respectively. The control voltages can be selected as:

$$u_d = u_{d0} sign(s_d)$$
$$u_q = u_{q0} sign(s_q) \qquad \textbf{8.10}$$

Once the control voltages u_d and u_q are obtained, the next step is to map them into the switching patterns of the inverter. Figure 8.2 and Figure 8.3 show the simulation results of the proposed speed controller. The control gains in equations 8.6 and 8.7 are designed as $c_1 = c_2 = 100$.

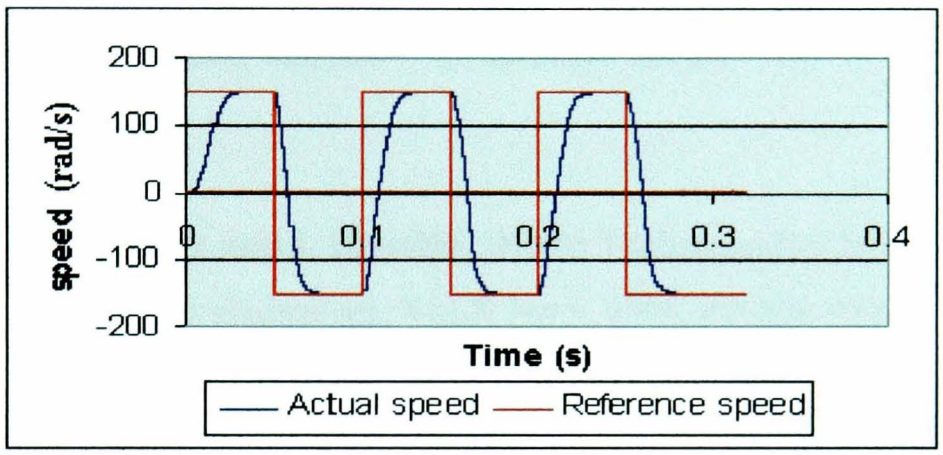


Figure 8.2 - *Response of the rotor speed control*

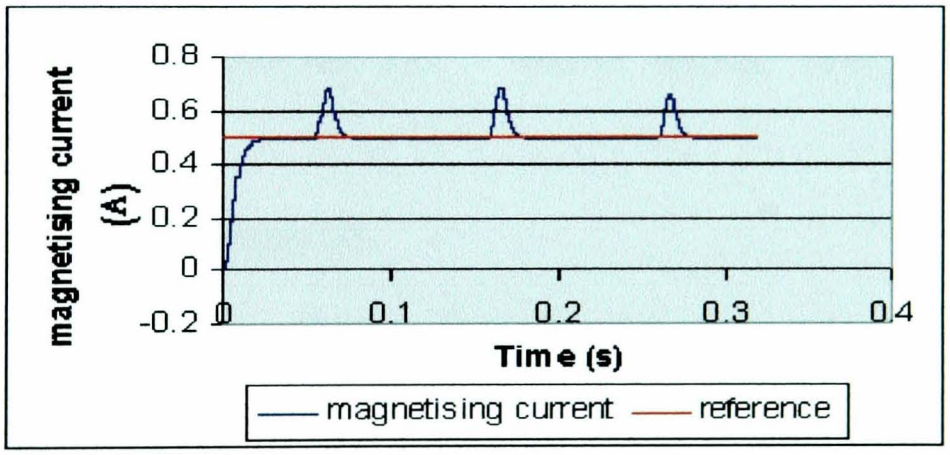


Figure 8.3 - *Response of the magnetizing current*

The advantages of this speed control method, using the sliding mode technique, are related to its robustness, high accuracy, fast dynamic response, good stability and simplicity of design and can be implemented with fewer numbers of gates and may require less CLBs when targeting the Spartan S40PQ208 FPGA. However, a proper investigation into this approach is required.

References

[1] **I. M. Gottlieb**, 'Electric Motors and Control Techniques', McGraw Hill, 1994.

[2] **R. M. Crowder**, 'Electric Drives and their Controls', Oxford University Press Inc, 1995.

[3] **F. Blaschke**, 'The Principle of Field Orientation as Applied to the New Closed Loop Control System for Rotating Field Machine', in siemense review Vol. 34, PP. 217-220, 1972.

[4] **T. Katsunori, O. Yasumasa and I. Hisaichi**, 'PWM Technique For Power MOSFET Inverter', IEEE Trans. on power electronics, Vol.3, No.3, 1998, PP.328-334.

[5] **M. David and W. Donald**, 'Current Control of VSI-PWM Inverters', IEEE Trans. on power electronics, vol. IA-21, No 4, May/June. 1985, PP. 562-570.

[6] **D. L. Perry,**'VHDL' McGraw-Hill Inc. - Second edition, 1994. (Computer Engineering Series).

[7] Foundation Series Software, Version 1.5.

[8] **M. H. Rashid-**'Power electronics- Circuits, Devices and Applications', Prentice-Hall Inc, 1988.

[9] **J. B. Baliga** –'The Insulated Gate Transistor (IGT) – A new power switching device' IEEE/IAS Ann. Meet. Conf. Rec., 1983, PP 794-843.

[10] **I. M. Gottlieb**.-' Electric motors and control techniques', McGraw Hill, inc 1994.

[11] **Xilinx**, The Programmable Gate Array Data Book, Xilinx Inc., San Jose, California, 1991.

[12] **Lucent Microelectronics**, Field Programmable System Chips, Eurodis Bytech Ltd, Basingstoke, England, 1998.

[13] **S. Charles**, User programmable gate arrays, EDN, Vol. 34, No. 9, Information Access, 27/04/89.

[14] **H. Robert**, The Softening of Hardware In Embedded Systems Design, Embedded Systems, 03/10/94 .

[15] **P. Ian**, Performance of a Fast GCD Program, Embedded solutions, Application Note 4, 15/05/98.

[16] **L. Kreindler, J.C. Moreira, A. Testa, T. A. Lipo**.,'Direct field orientation controller using stator phase voltage third harmonic'. IEEE Trans Ind. Apll. Vol. 30, No. 2, 1998, PP.441-1994.

[17] **B. K. Bose.**' High performance control of induction motor drive', UK @http//sant.bradley.edu/~ienews/98_3/bose.htm.

[18] **D. M. Bord and D. W. Novotny.-**'Current control of VSI PWM inverters', IEEE Trans. Ind. Appl., Vol. IA-21, No. 4, PP.562-570, 1985.

[19] **A. B. Plunkett.-**' A current-controlled PWM transistor inverter drive', conference. Rec. 1979 14th Annu. Meet. IEEE Ind. Appl. Soc., PP 785-792.

[20] **A. Nabae, S. Ogasawara, and H. Akagi.-**'A novel control scheme for current controlled PWM inverters', IEEE Trans. Ind. Appl. vol IA-22, No. 4, July/August 1986, PP.697-701.

[21] **J. Holtz and S. Stadtfeld.-**' A PWM inverter drive system with on line optimized pulse patterns', EPE Conf.Rec.,Brussels 1985, PP.3.21-3.25.

[22] **H. Le-Huy and L. A. Dessaint.**-' An adaptive current control scheme for PWM synchronous motor drives: analysis and simulation', IEEE Trans on power electronic vol. 4, No. 4, October 1998, PP.486-495.

[23] **M. P. Kazmierkowski , M. A. Dzieniakowski and S. Waldemar.**-' Novel space vector based current controllers for PWM inverter', IEEE Trans on power electronic vol. 6, No. 1, January 1991, PP.158-165.

[24] **B. H. Kwon, T. W. Kim and J. H. Youm.**-' A novel SVM based hysteresis current controller ', IEEE Trans on power electronic vol. 13, No. 2, March 1998, PP.297-307.

[25] **M. Tsuji and E. Yamada**.- 'Advanced vector control for induction motor drive', Sppdam symposium on power electronics, 3-5 June, 1998 (Italy), PP A1-1 to A1-7.

[26] **P. Vas**, 'Vector Control of AC Machine.-, Clarendon Press, Oxford, UK, 1990.

[27] **K. Ohyama, G. M. Asher and M. Sumner**.-' Comparision of the practical performance and operating limits of sensorless induction motor drive using a closed loop flux observer and a full order flux observer', EPE 99-Lausanne, PP 1-10.

[28] **A. Damiano, P. Vas, I. Marongiu**.-,' Comparision of speed sensorless DTC induction motor drives', Intelligent motion, June1997 proceeding, PP 1-11.

[29] **Y. Hwa Liu and C. Chen**.-' A novel regulation scheme for DTC', Intelligent motion, June1998 proceeding, PP 269-273.

[30] **F. Moldoveanu, I. Topa, V. Commac and D. Floroian**.-' Nonlinear torque control of an inverter-fed asynchronous motor', Speedam Symposiun on power electronic, 3-5 June 1998 (Italy), PP p3-1 to 3-5.

[31] **F.Aubepart, C. Girerd, Y. A. Chapuis, P. Poure and F. Braun**.-' ASIC implementation of direct torque control for induction machine: functional validation by power and control simulation', Inteeligent motion, May 1998, PP 251-260.

[32] **A. M. Arcker-Hissel, H. Schneider and M. Pietrzak-David**.-' Real time analog technology performing variable structure control of high power induction motor drives', in proceedings of (Speedam'98) Sorrento (Italy), June 1998, PP C2-1-C2-6.

[33] **K. Shyu and H. Shieh**.-' A new switching surface sliding mode speed control for induction motor drive system', IEEE Trans on power electronics. Vol. 11, No. 4, July 1996, PP.660-666.

[34] **N. Mdani, M. F. Benkhoris, s. S.Siala and M. O. Mahmoudi**.-' Sliding mode control of an asynchronous motor drive', Power electronics and variable speed drive, 21-23 September 1998, PP 341-346.

[35].**A. Benchiab, A. Rachid and E. Audrezet**.-' Sliding mode input-output linearization and field orientation for real time control of induction motors' IEEE Trans on power electronics. Vol. 14, No. 1, January 1999, PP.1-13.

[36] **J. Fernando Silva**.-' Sliding mode control of boost type unity power factor PWM rectifiers' IEEE Trans on industrial electronics. Vol. 46, No. 3, June 1999, PP.594-603.

[37] **Y. R. Kim, S. K. Sul, M. H. Park**.-' Speed sensorless vector control of induction motor using extended kalman filter' in IEEE Trans. Ind. Appl., Vol. 30, No. 5, PP. 1225, 1994.

[38] **C. Manes, F. Parasiliti, M. Tursini**.- ' DSP based field-oriented control of induction motor with a non linear state observer' in 27th Annual IEEE power Electronics Specialists Conference, Vol. 2, PP.1254, 1996.

[39] **V. Navrapescu, A. Craciunescu and M. Popescu**.-' Discrete- time induction machine mathematical model for a DSP implementation', in Intelligent motion proceedings, May 1998, PP 433-438.

[40] **J. F. Moynihan, P. Kettle and A. Murray**.-' High performance control od aAC servomotors using an integrated DSP', in Intelligent motion proceedings, May 1998, PP 213-222.

[41] **L. Zhen and L.Xu**.-' On line fuzzy tuning if indirect field oriented induction machine drives', IEEE Trans on power electronics. Vol. 13, No. 1, January 1998, PP.134-138.

[42] **H. C Tseng and V. H. Hwang**.-' Servocontroller tuning with fuzzy logic', IEEE Trans on power electronics. Vol. 4, No. 4, December 1993, PP.262-269.

[43] **J. H. Kim, K. C. Choon and K. P. Chong**.-' Fuzzy precompensated PID controllers', IEEE Trans on power electronics. Vol. 2, No. 4, December 1994, PP.406-411.

[44] **N. B. Mrabet, K. Jelassi, L. Constant and B. Dagues**.-' Comparative study of classical estimator and neural estimator for induction machine flux' in proceedings of (Speedam'98) Sorrento (Italy), June 1998, PP B1-1-B1-5.

[45] **W. P. Hew, M. R. Tamjis and S. M. Saddique**.-' A computer based current sensorless induction motor drive', in proceedings of (Speedam'98) Sorrento (Italy), June 1998, PP P3-13-P3-16.

[46] **A. Dinu, M. N. Cirstea, M. McCormick, A. Ometto and N. Rotonadale**.-' Load independent adaptive current control strategy for PWM inverters', in proceedings of

UKACC , International Conference on Control (Control 98) september 1-4 1998 Swansea Uk, PP 1118-1122.

[47] **A. Dinu**.- 'FPGA Neural controller for three-phase-sensorless induction motor drive systems ', PhD Thesis, De Montfort University , June 2000.

[48] **M. N. Cirstea, A. Dinu, J. Khor, M. McCormick**, ' Neural and fuzzy logic control of drives and power systems', Elsevier Science Ltd, Oxford, 2002, UK, ISBN 0750655585.

[49] **E. B. Patterson, P. G. Holmes and D. Morley**, ' Electronic design automation (EDA) techniques for the design of power electronic control systems' IEE proceedings-G, Vol. 139, No. 2,April 1992.

[50] **M. N. Cirstea**, ' An investigation into ASIC control of a 6-pulse cycloconverter for quadwinding induction motor', Electric and Electronic Engineering, Nottingham Trent University, UK, 1996.

[51] **T. Y, J. Hau**, ' FPGA Realization of space vector PWM control IC for three phase PWM inverters' IEEE Trans on power electronic, Vol. 12, No. 6, November 1997.

[52] **V. K. Madisetti.** (1995). Digital signal processors, (IEEE Press, Butterworth Heinemann).

[53] **S. Yalamanchili.** (1998). VHDL Starter's guide (Prentice-Hall Inc).

[54] **J. V. Oldfield and R. C. Dorf,** 'Field programmable gate arrays'.John Wiley and Sons, inc,1995.

[55] **T. Riesgo, Y. Torroja and E. Torre,** 'Design methodologies based on hardware description language' IEEE Trans on industrial electronics. Vol. 46, No. 1, Feb 1999.

[56] **J. G. Khor**.- 'An intelligent controller for synchronous generators ', PhD Thesis, De Montfort University , Nov 1999.

[57] **K. Bartleson, '**VHDL emerges as a commercial design tool', United Technologies Microelectronics Center. Aug 1994.

[58] **P. Vas,** 'Vector Control of AC Machine', Clarendon Press, Oxford, UK, 1990.

[59] **W. Scott, W. Matthew and W. Barry,** 'Modeling and Simulation of Induction Machine Vector Control with Rotor Resistance Identification', IEEE Trans. on power electronics, vol.12, May. 1997, PP. 495-503.

[60] **Z. Navabi.** 'VHDL: Analysis and Modeling of Digital Systems', McGraw Hill,1998.

[61] **H. John,** 'Numerical Methods for Mathematics, Science and Engineering', Prentice Hall, 1992.

[62] **F. Zdenek, K. Karel and P. Miroslav,** 'Digital Signal Processor Application for Vector Control of Asynchronous Motor', Speedam, Sorrento (Italy), June 1998, PP. P3-7 P3-11.

[63] **Texas Instruments,** 'DSP Solution for a.c. Induction Motor', 1996.

[64] **K. Tazi and E. Monmasson,** 'Single-Chip DSP Based Speed Control of Two AC-Machine', Speedam, Sorrento (Italy),3-5 june 1998, PP. P4-33 to P4-38.

[65] **P. Krafka , A. Bunte and H. Grotstollen,** 'Comparison of Induction Machine Control With Orientation on Rotor Flux in a Very Wide Field Weaking Region. EPE, Sevilla, 1995.

[66] **F. Aubepart, C. Girerd, Y.chapuis, P.poure and F. Braun,** 'ASIC implementation of direct torque control for induction machine functional validation by power and control simulation' intelligent motion. May 1998. PP 251-260.

[67] **K. Tazi and E. Monmasson,** 'Single chip DSP based speed control of two AC-machine ' Speedam. 3-5 June 1998 (Italy). PP P4-33-39.

[68] **DSP** solution for a.c. induction motor- Application report,© 1996, Texas instruments.

[69] Foundation series software, version 1.5.

[70] **Richards**. (1971). Degital design (Wiley Interscience).

[71] Texas instruments," Field Oriented Control of 3- Phase AC-Motor"-©, 1998.

[72] **Y. Kawabata. Y. Nishiyama, H. Asai, E. Ejioqu and T. Kawabata**,' High efficiency drive system using three level inverter and two level inverter', EPE, Lausanne, Switzerland, 1999.

[73] **B.Robyns, R. Meuret, P. Sente**,'Current digital control influence on performance of induction motor indirect field oriented control', EPE, Lausanne, Switzerland, 1999.

[74] **R. Mecke, U. Riefenstahl**,' A new topology for PWM current source inverter with IGBT for high power', EPE, Lausann,1999.

[75] **S. Vadivel, G. Bhuvaneswari, S. G. Rao**, ' A unified approach to real time implementation of DSP based PWM waveforms', IEEE Trans. Power Electronics. Vol. 6, No. 4, PP. 565-575, 1991.

[76] "Foundation Series Software, Xilinx Stuent Edition 1.5", Prentice Hall

[77] **M. Cirstea, A. Dinu, D. Nicula**, 'A Practical Guide to VHDL Design', Editura Thnica, 2001, ISBN 973-31-1539-8.

[78] *** "FH2/3 MKIV Electrical Machines Teaching System". TecQuipment, Bonsall Street Long Eaton, Nottingham, NG102AN, UK, 1997.

[79] *** "High Speed Programmable Gain Instrumentation Amplifier", Burr-Brown Corporation, 1994.

[80] **J. Khor**.- 'An Intelligent Controller for Synchronous Generators', PhD Thesis, De Montfort University, November 1999.

[81] **V. Utkin, J. Guldner, J. Shi** – ' Sliding Mode Control in Electromechanical Systems', Taylor and Francis Press, ISBN 0-7484-0116-4, 1999.

LIST OF PUBLICATIONS

- V. Comnac, M. McCormick, A. Aounis, M. Cernat, R. M. Cernat - " The Control of Interior Permanent Magnet Synchronous Motor using a Non-Linear Minimum Order Observer." Proc. Of 8th Int. Conf. On Power Electronics and Applications (EPE99), EPFL Lausanne, Switzerland, 1999. pp P.1-P.9.

- M. Cirstea, A. Aounis, A. Dinu, M. McCormick - " A VHDL Approach to Induction Motor Modelling." Proc. Of the IEE Control 2000, Cambridge, UK, 4-7 September 2000, CDROM.

- A. Aounis, M. Cirstea, M. McCormick, A. Dinu - " Vector Controlled Induction Motor Drive Modelling Using VHDL." Proc. Of IEE Int. Conf. on Computer Aided Control Systems Design (CACSD), Salford, UK, 11-13 Sept. 2000, CDROM.

- M. Cirstea, A. Aounis, M. McCormick, P. Urwin, L. Haydock - " Induction Motor Drive System Modelled in VHDL " , in the Proceeding of the IEEE International Conference on VHDL International Users Forum Fall Workshop, Orlando, Florida, 18-20 October 2000,pp.113-117

- M. Cirstea, A. Aounis, M. McCormick, P. Urwin, L. Haydock - " Vector Control System Design and Analysis Using VHDL ", in the 23rd IEEE International Conference on Power Electronics Specialists, Vancouver, Canada, 17-22 October 2001, CDROM.

- A. Dinu, M. Cirstea, M. McCormick, A. Aounis - " Sensorless Induction Motor FPGA Controller Using Polar Coordinates ", in The Proceeding of International Conference on Drives and Controls and Power Electronics, , London, 13-15 March 2001pp 19-24.

- A. Aounis, M. McCormick, M. Cirstea - " A Novel Approach to Induction Motor Controller Design and Implementation ", in the IEEE Proceeding of the Power Conversion Conference, Osaka 2002,Vol III, pp 993-998.

- M. Cirstea, A. Aounis, M. McCormick - " Rapid Prototyping of Induction Motor Vector Control System Based on Reusable VHDL Digital Architecture and FPGA Implementation", in The Proceeding of International Conference on Power Electronics Intelligent Motion Power Quality (PCIM), Germany, 12-16 May 2002. CDROM

Paper Accepted for Publication:

- A. Aounis, M. McCormick, M. Cirstea - " VHDL Modelling, Simulation and Implementation of A Vector Controlled Drive", Proc. Of IEEE Int. Conf. On Power Electronics, Cairns, Australia, 23-27 June.2002.

PCIM Europe 2002

45. International Conference
Power Electronics / Intelligent Motion / Power Quality, Nuremberg, May 14 - 16, 2002

A Resolution of Appreciation

TO

A. Aounis

OF

De Montfort University, UK

FOR PRESENTING

RAPID PROTOTYPING OF INDUCTION MOTOR VECTOR CONTROL SYSTEM BASED ON REUSABLE VHDL DIGITAL ARCHITECTURES AND FPGA IMPLEMENTATION

Be it therefore resolved that the Conference Directors shall express, on behalf of PCIM 2002 Europe, its deep appreciation for this outstanding contribution to the advancement of the industry´s technology.

Jean-Marie Peter, France
General Conference Director

Prof. Helmut Knöll, University of Applied Sciences
Wuerzburg-Schweinfurt, Germany
Technical Director Intelligent Motion

Organizer: ZM Communications GmbH

Proceedings of the Power Conversion Conference-Osaka 2002

Volume III

PCC OSAKA 2002
Power Conversion Conference-Osaka

sponsored by

the Industry Applications Society of the IEE Japan
and
the IEEE Industry Applications Society

in cooperation with

the IEEE Industrial Electronics Society,
the IEEE Power Electronics Society,
IEEE Japan Council,
the China Electrotechnical Society,
Japan Society for Power Electronics,
the Korean Institute of Electrical Engineers,
the Society of Instrument and Control Engineers,
the Illuminating Engineering Institute of Japan,
the European Power Electronics and Drives Association,
the Institute of Electrical Installation Engineers of Japan,
the Institute of Systems, Control and Information Engineers,
the Institute of Electronics, Information and Communication Engineers.

IEEE Catalog Number 02TH8579

A Novel Approach to Induction Motor Controller Design and Implementation

A. Aounis, M.McCormick, M. Cirstea

Electronic and Electrical Engineering Department,
De Montfort University, Leicester, UK
E. mail: Aounis@yahoo.com

Abstract

The paper presents a new approach to the modelling, simulation, digital controller design and implementation of a complete induction motor electric drive system. The design approach is to uses a very high speed integrated circuit hardware description language (VHDL) as a unique EDA environment for system modelling, evaluation and controller design. Simulation and experimental results are presented, proving the validity of the novel approach when applied to a vector controlled induction motor system. The advantages of using VHDL for a drive modelling and control strategy implementation are enumerated.

Key words: Induction motor, control, VHDL modeling

1. Introduction

It has been estimated that induction motors are used in seventy to eighty percent of all industrial drive applications due to their simple mechanical construction, low maintenance requirement and lower cost compared to other types of motors, such as brushless d.c. motors [1][2]. General research in the area of induction motors control is characterised by a large number of control methodologies with different control observation and adaptive algorithms, combined with different coordinate systems, state variables and notations [3][4][5].
From a control view point, induction motors are non linear high order systems of considerable complexity. Due to the highly sophisticated mathematical content of the equations, the signal processing performed by the control systems is complicated and expensive.
Vector control techniques for a.c. drives demand accurate position and speed feedback information for the current control and servo-control loops. In many applications, the rotor position signal is obtained from a mechanical sensor, such as an optical encoder or a resolver, that may reduce the reliability of the system and significantly add to the drive costs. Consequently, there is a strong interest in sensorless control, based on state observers using only stator voltage and current measurement. State observers are usually implemented to reconstruct the inaccessible states of the controlled process. They are especially useful for full-state feedback control, developed using state-space theory, as the combined observer-controller system can easily be designed to meet specific qualitative and quantitative requirements.
Although most a.c. drives in use today adopt a microprocessor based digital control strategy, implementation of current loop and PWM control is still tied to analog control circuitry. This kind of control scheme has the advantages of fast dynamic response but suffers the disadvantage of circuit complexity, limited functions and difficulties when circuit modification is required[6].
DSP technology has been generally used for digital a.c. motor control implementation [7]. However recent developments in the field of microelectronics have enabled complex switching strategies for transistorized inverters to be implemented by means of ASIC technology. Consequently motor control and power conversion systems employing ASIC/FPGA technology are receiving increased attention .
This paper presents a comprehensive study of the integrated modelling, simulation and design of a vector controlled induction motor using VHDL hardware description language and targeting FPGA implementation. The top-down methodology consists of modelling the system hierarchically at all levels of abstraction, ranging from the behavioural level to the logic gate level.

2. System modelling

In order to achieve the performance required by servo applications, induction motors are controlled using vector control strategies. The drive system model outlined in Fig. 1, allows high performance control of torque, speed or position to be achieved.

The complete drive system can be modelled, simulated and evaluated using Very High Speed Integrated Circuit Hardware Description Language (VHDL). This is now one of the most popular standard HDLs, comprehensively described in [8][9]. It is supported by all major Computer Aided Engineering (CAE) platforms, in addition synthesis tools can compile VHDL designs into a large variety of target technologies. The actual design of the digital controller was achieved in VHDL, and was then synthesised and downloaded into a XILINX FPGA for rapid prototyping. The VHDL approach presented in this paper provides important advantages such as: wide compatibility of the design with respect to different CAE software tools (VHDL files are ASCII files), a large range of implementation technologies and the reuse of the VHDL code.

FPGAs are implemented with the regular flexible programmable architecture of Configurable Logic Blocks (CLBs) interconnected by a powerful hierarchy of versatile routing resources (routing channel) and surrounded by a perimeter of programmable Input/ Output Blocks. They have generous routing resources to accommodate the most complex interconnection pattern [10]. The devices are customised by loading configuration data into internal static memory cells. Reprogramming is possible an unlimited number of times. FPGAs are ideal for shortening design and development cycles, and also offer a cost effective solution enabling a higher production rates to be attained.

Although VHDL is a hardware description language and as such is primarily used for circuit design, it has the basic properties of any software programming language. This allows complete power system modelling and simulation, prior to digital controller design and implementation.

To verify the model, a simulation using VHDL is carried out with an integration step size of (0.000001) using Euler's method.

Fig. 2 shows typical speed versus time plots illustrates the tracking of the actual speed to the reference speed.

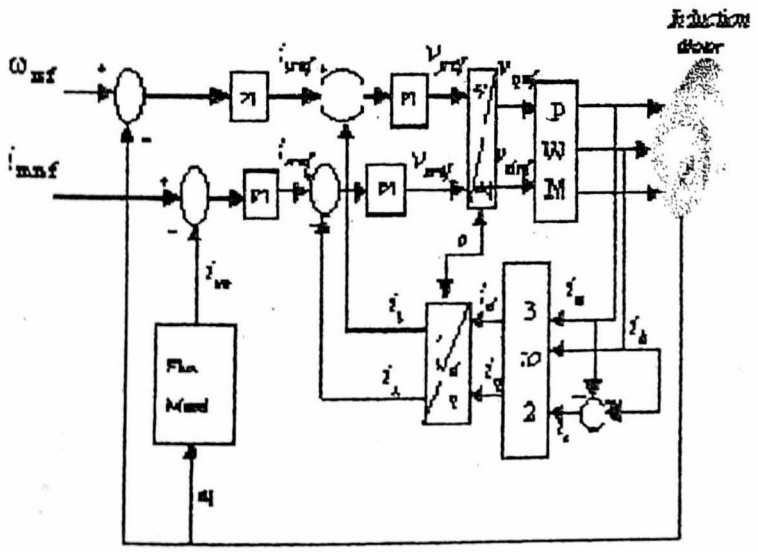


Fig 1 - Control principle of a.c. machine.

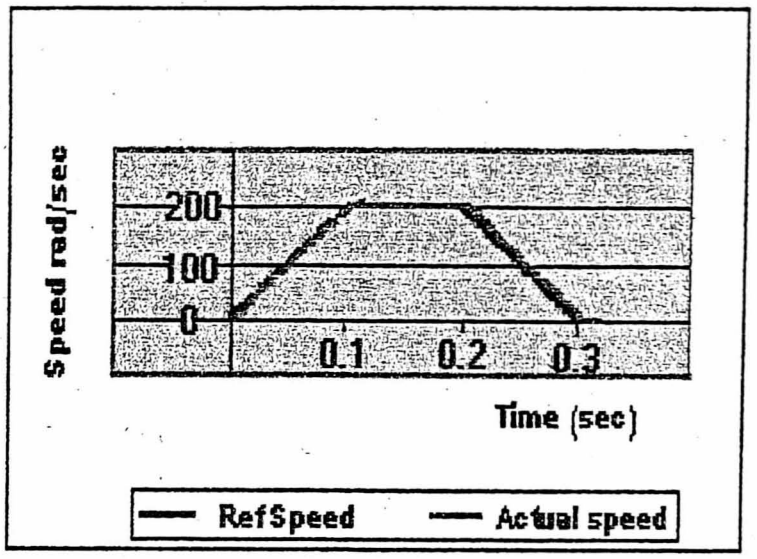


Fig2 - Reference and actual rotor speed.

An important aspect in modeling the control system is the representation of the machine using the Clark and Park transforms. Their representation in VHDL is described followed by implementation in FPGA technology. The PWM waveform generator and performance characteristics are then demonstrated and the overall strategy is validated by the results obtained.

3.0 Clark transform

The Clark transform is based on the principle of using an intermediate coordinate system α, β and its phase current projections i_α and i_β. Therefore:

$$i_\alpha = i_a \qquad (1)$$

$$i_\beta = \frac{1}{\sqrt{3}}(2 * i_b + i_a) \qquad (2)$$

In order to solve equation (2), an integer approximation of the $1/\sqrt{3}$ is expressed as:

$$\frac{1}{\sqrt{3}} = 0.577 = 0.100100111011 \text{ in binary.}$$

Digital representation in 2 complementary of 0.577 using VHDL can be achieved by considering either the 1's or the 0's. If 1's are considered, then 0.577 can be approximated by:

$$\frac{1}{\sqrt{3}} = \frac{1}{2^1} + \frac{1}{2^4} + \frac{1}{2^7} + \frac{1}{2^8} + \frac{1}{2^9} + \frac{1}{2^{11}} + \frac{1}{2^{12}} \approx$$

$\approx 0.5+0.0625+0.00781+0.0039+0.0019+0.000488+0.000244$

$\approx$ **0.5769**

If 0's are considered, then 0.577 can be approximated by:

$$\frac{1}{\sqrt{3}} = 0.577 \approx 1 - \left(\frac{1}{2^2} + \frac{1}{2^3} + \frac{1}{2^5} + \frac{1}{2^6} + \frac{1}{2^{10}}\right) \approx$$

$$\approx 1 - 0.25 + 0.125 + 0.03125 + 0.015625 + 0.000976)$$
$$\approx \underline{\mathbf{0.57714}}$$

The second method has been chosen and is represented in VHDL as shown in Fig. 3. Fig. 4 shows the simulation results of Clark transform.

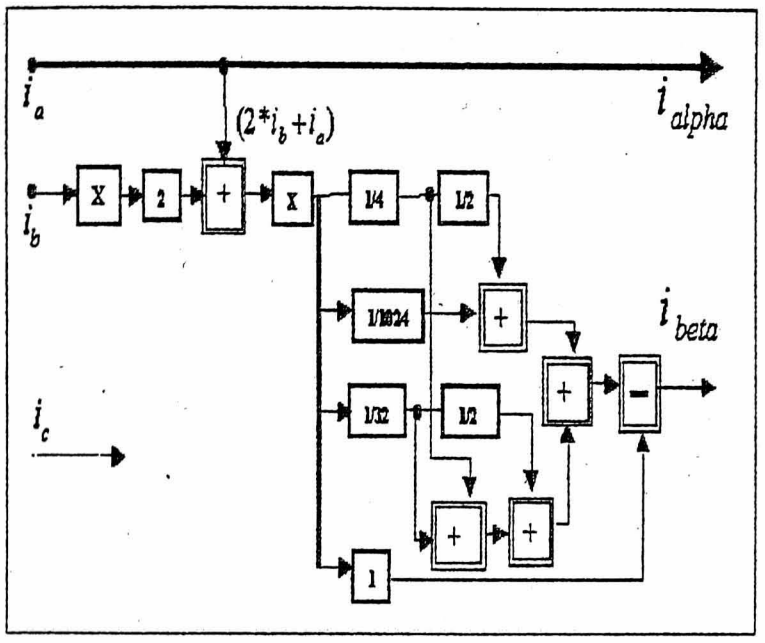


Fig 3 – the 9 bit realization of the Clark transform.

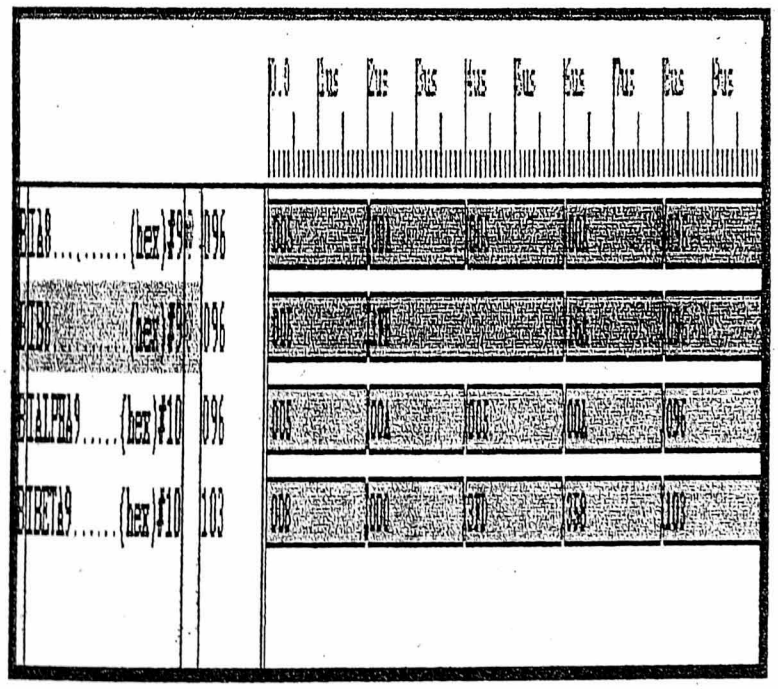


Fig 4 – Simulated waveforms for Clark transform.

4.0 Park transform

The Park transform is the most complex transformation in the Field Oriented Control system (FOC). The two phases (α, β) frame representation calculated within the Clark transform is fed to a vector rotation block where it is rotated through an angle ϑ to follow the frame (d, q) attached to the rotor flux. The rotation over an angle ϑ is according to the formulas:

$$i_{sd} = i_{alpha} * \cos(teta) + i_{beta} * \sin(teta) \qquad (1)$$

$$i_{sq} = -i_{alpha} * \sin(teta) + i_{beta} * \cos(teta) \qquad (2)$$

Before a VHDL design can be synthesised all VHDL functions which are not synthesisable have to be eliminated and replaced with functions that can be translated into hardware. It is obvious from the equations that the Park transform requires several multiplications, additions and subtractions.

The VHDL description of the Park transform is represented in Fig. 5 as a block diagram. Each of the 8 VHDL components is depicted with its VHDL code file '<filename>'.vhd'. The VHDL structure of the Park transformation consists of:

- ➔ Control unit.
- ➔ Multiplier.
- ➔ Register1, Register2, Register3 and Register4.
- ➔ Adder and Subtracter.

Each element of the Park transform is designed and optimised for synthesis. Fig. 7 and Fig. 9 show the experimental results of the test performed by downloading the Park transform component design to the Xilinx Spartan series of FPGA. Only the first four bits of i_d and i_q are plotted to be used as a comparison between the implemented design and the simulated one of Fig. 6 and Fig. 8. It can be seen from the practical tests that the implemented design generates the same values of i_d and i_q for the first four LSBs as the simulated values highlighted by the triangular box.

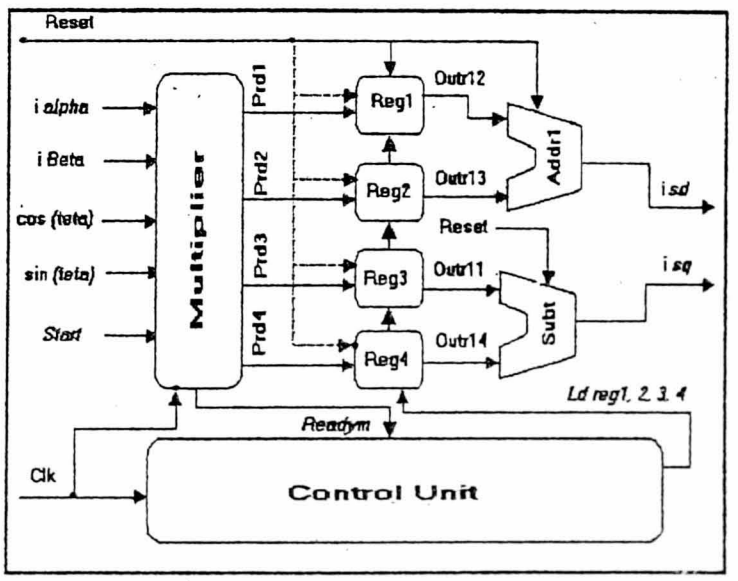


Fig 5 - Block diagram of Park transform.

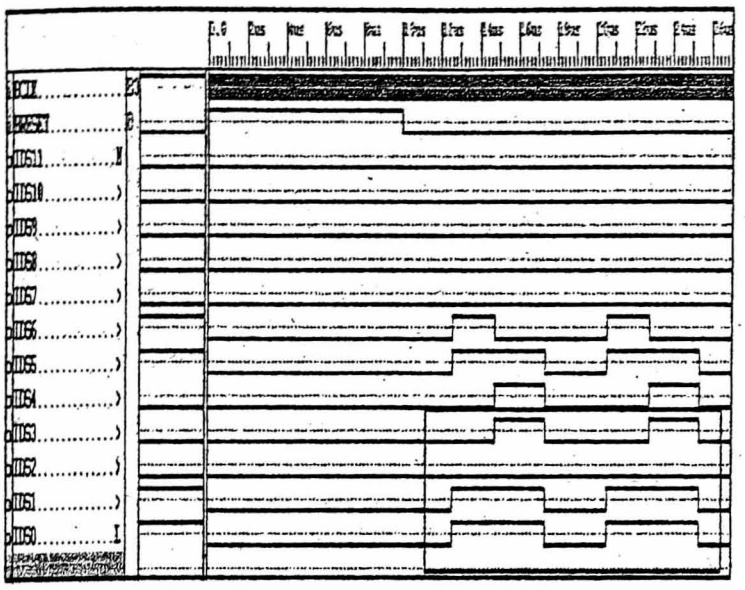


Fig 6 - waveforms of i_d for the simulated circuit of the Park transform.

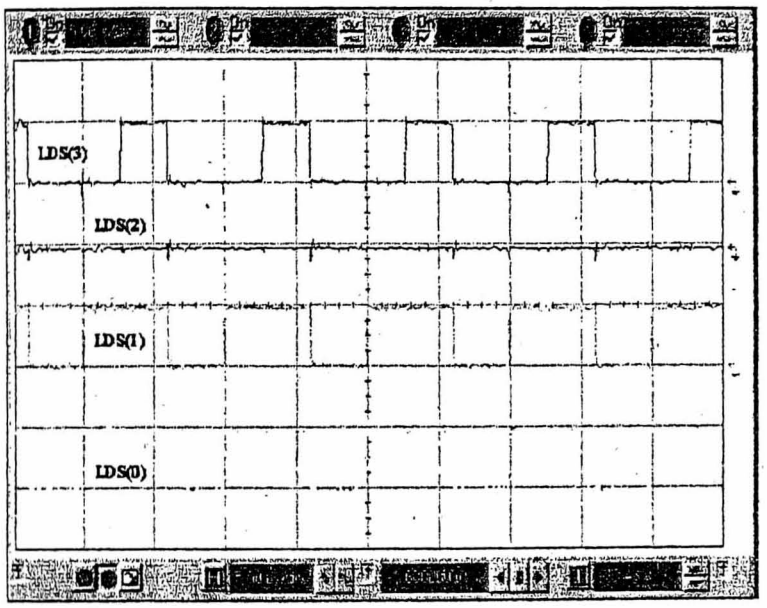


Fig 7 - waveforms of i_d for the implemented circuit of the Park transform.

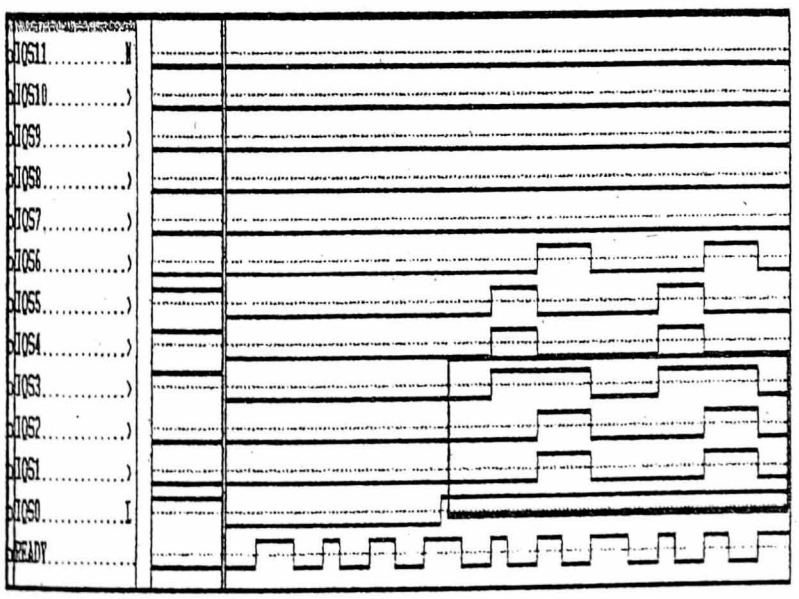


Fig 8 – waveform of i_q for the simulated circuit of the Park transform.

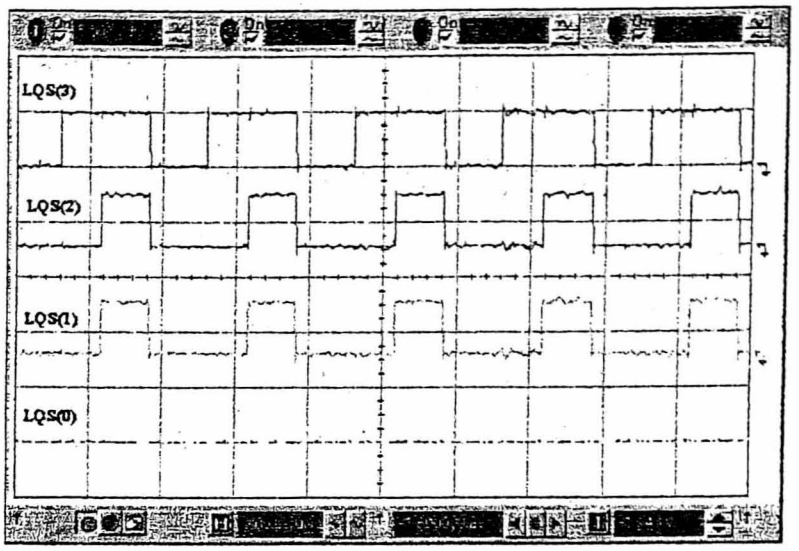


Fig 9 – waveform of i_q for the implemented circuit of the Park transform.

5.0 HARDWARE IMPLEMENTATION AND PRACTICAL TESTS

FPGAs are ideal for shortening the design and development cycles and they allow the exploitation of the parallel processing enabled by hardware implementation. The presented control strategy was implemented in Xilinx Spartan XCS40PQ208 FPGA containing the equivalent of 40000 logic gates. The device has a total of 784 CLBs The hardware structure obtained after the synthesis covers 99% of the chip and it operates at a clock frequency of 12 MHz. Fig. 10 shows the hierarchy of individual subcomponents (subdesign entities) for the Field Oriented Control, brought together in one higher level component (design entity). This work is based on the implementation process required to convert the netlist into a format that can be downloaded into the target technology.

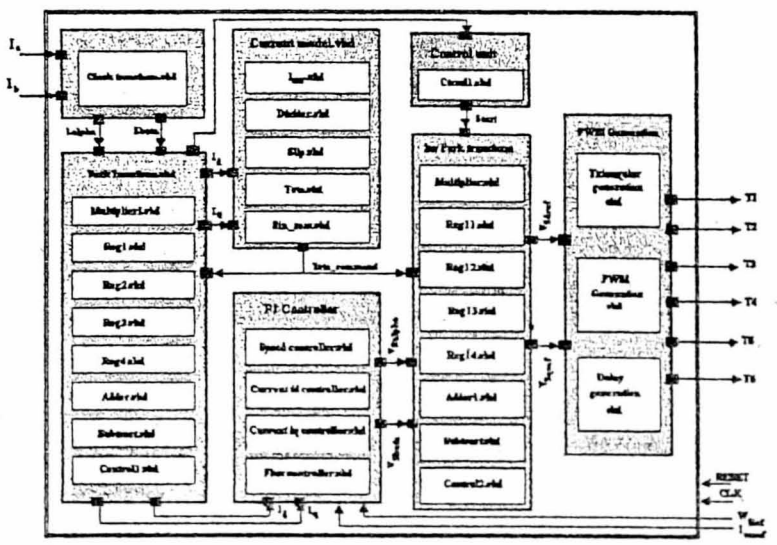


Fig 10 - VHDL Hierarchy.

5.1 PWM waveform generator

Recent developments in the field of microelectronics have enabled complex switching strategies for transistorized inverters to be implemented by means of ASIC technology. The structure of the PWM waveform generator circuit adopted is illustrated in Fig. 11.

The PWM generator operation is controlled by means of clk and clkr signals alongside the values supplied through the maximum and minimum values external bus. The maximum count values determine the amplitude of the triangular carrier generated by the reversible up-down counter. The counter generates successive numbers between min count and max count in increasing and decreasing order, thereby producing the digital equivalent of the triangular carrier signal used by analogue PWM generators. Consequently the period of the digital carrier will be proportional to its amplitude.

The VHDL model design is divided into two processes. One is used to describe the operational logic of the triangular waveform generator process unit whereas the other process is used to generate the correct signal to control the transistors in the PWM inverter. The output signals defining the desired output voltage of the PWM inverter are transformed into six logic signals.

Each signal is amplified and applied to one power transistor. The edges of these signals are shifted by approximately 5 μs so that short circuits are avoided between transistors on the same inverter phase (*T1* and *T2* for example) as illustrated in Fig. 12 which clearly indicates the delay time between the two signals. Fig. 13 presents four of the PWM control signals generated by the FPGA motor controller. They have been monitored using a four channel Hewlett Packard digital oscilloscope. The figure demonstrates the correlation between two of the signals that control the transistors on the same inverter leg *T1* and *T2*. The two signals have complementary values: when one of them is '0' the other is '1' as illustrated in Fig. 14.

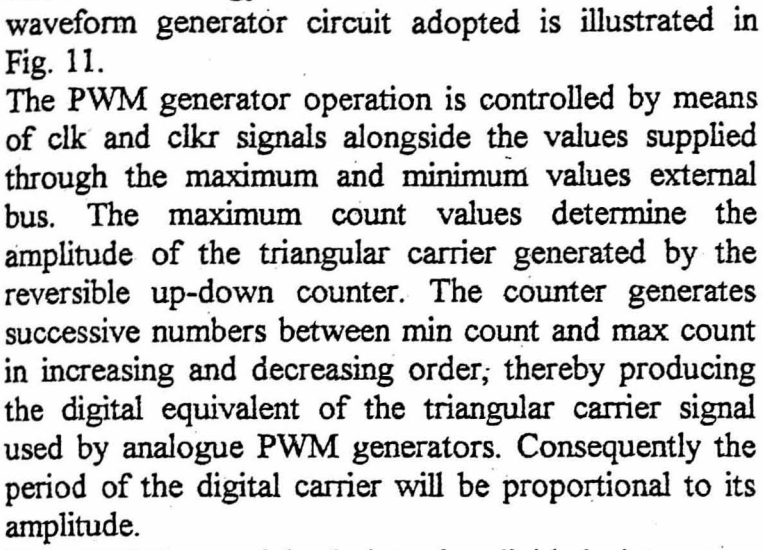


Fig 11 - PWM wave generator.

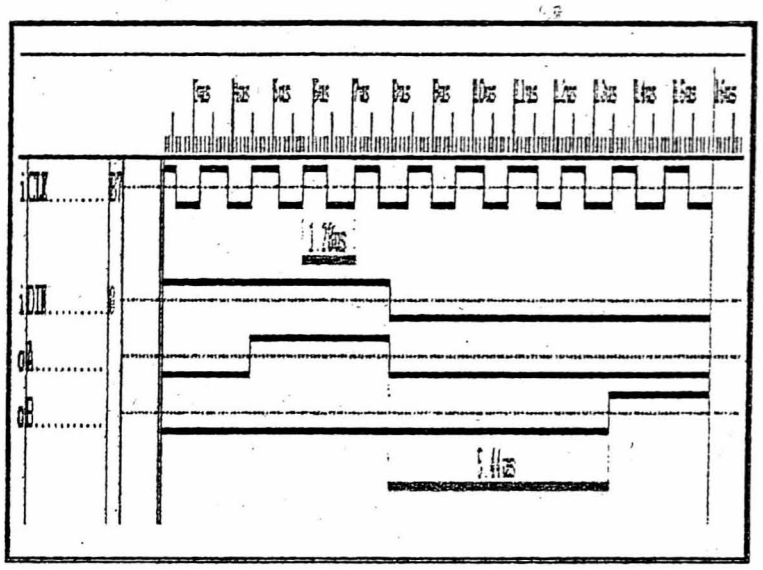


Fig 12 – Simulated waveform showing the delay time.

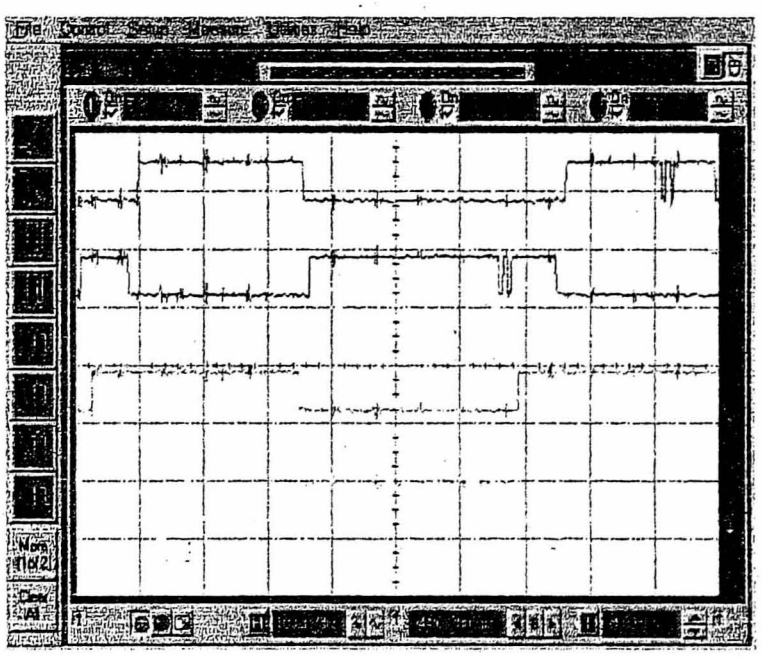


Fig 13 - Four of the FPGA output signals controlling the transistors in the PWM inverter.

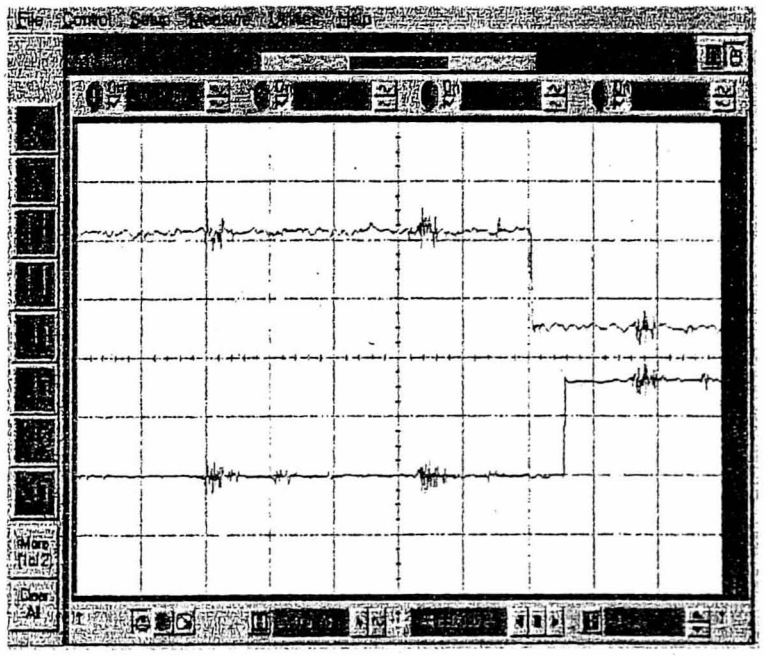


Fig 14 - The switching delays produced by the FPGA to avoid the short circuits in the PWM inverter.

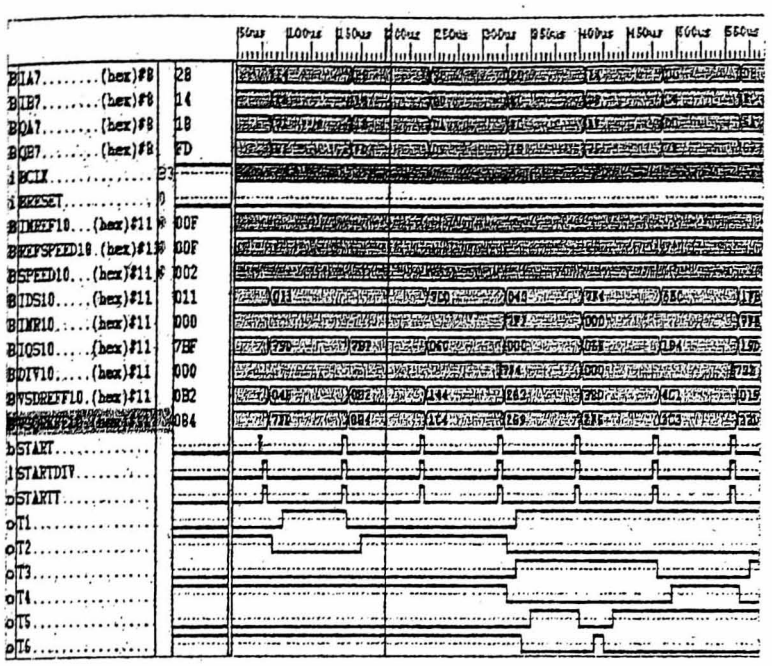


Fig 15 - Overall simulation of the induction motor controller

The overall simulation of the induction motor controller is illustrated in Fig. 15, where i_a, i_b, i_{mref} and *ref_speed* are used as inputs and the switching patterns of the 6 power transistors *T1, T2, T3, T4, T5* and *T6* are the output of the FOC.

6.0 Conclusion

A new approach has been developed for the modelling and design of a complete vector controlled induction motor drive system. The simulation results of the VHDL model have been presented, showing the expected behaviour of the motor model. The paper also presents a new design and implementation of the Park transform and the PWM control modules for the induction motor. These are implemented employing FPGA technology. The target technology for this design is the Spartan XCS40/XL, a member of the Xilinx Spartan series of FPGA.There are major advantages of the new approach, such as:

- A unique environment for modelling, simulation and evaluation of complete drive systems, including controllers, power electronics and induction motors.
- The same environment (VHDL) is used for the design of the digital controller achieving the vector control strategy and for silicon (FPGA) implementation.
- Fast design development and short time to market.
- A CAD platform independent model and design are developed (VHDL operates with ASCII files) and therefore valuable IP can be produced, in co-relation with the modern principles of design reuse.
- Concurrent engineering basic rules (unique EDA environment and common design database) are fulfilled.

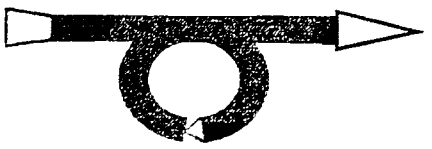
The complete digital controller circuit design is currently being synthesised and will be subsequently reported. XILINX FPGA implementation is targeted.
The extensive use of VHDL for drive systems modelling and simulation, in conjunction with accurate digital controllers design is foreseen as a practicality in the near future.

7.0 Reference

[1] I. M. Gottlieb, 'Electric Motors and Control Techniques', McGraw Hill, 1994.
[2] R. M. Crowder, 'Electric Drives and their Controls', Oxford University Press Inc, 1995.
[3] T. Katsunori, O. Yasumasa and I. Hisaichi, 'PWM Technique For Power MOSFET Inverter', IEEE Trans. on power electronics, Vol.3, No.3, 1998, pp.328-334.
[4] M. David and W. Donald, 'Current Control of VSI-PWM Inverters', IEEE Trans. on power electronics, vol. IA-21, No 4, May/June. 1985, pp. 562-570.
[5] A. Nabae, S. Ogasaward and H. Akagi, 'A Novel Control Scheme for Current Controlled PWM Inverters', IEEE Trans. on power electronics, vol. IA-22, No 4, July/August. 1986, pp. 697-701.
[6] D. Perry, - "VHDL", McGraw-Hill, 1998.
[7] Z. Navabi. 'VHDL: Analysis and Modeling of Digital Systems', McGraw Hill,1998.
[8] K. Tazi and E. Monmasson, 'Single-Chip DSP Based Speed Control Of Two AC-Machine', Speedam, Sorrento (Italy),3-5 june 1998, pp. P4-33 to P4-38.
[9] P. Krafka and A. Bunte and H. Grotstollen, 'Comparison of Induction Machine Control With Orientation on Rotor Flux in a Very Wide Field Weaking Region. EPE, Sevilla, 1995.
[10] P. Vas, 'Vector Control of AC Machine', Clarendon Press, Oxford, UK, 1990.

8TH IFAC SYMPOSIUM

IFAC



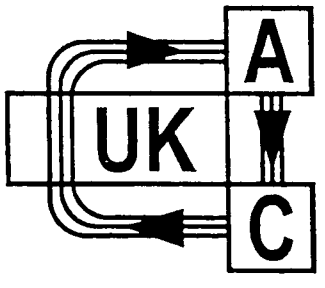
CACSD 2000

COMPUTER AIDED CONTROL SYSTEMS DESIGN

11-13 SEPTEMBER 2000

UNIVERSITY OF SALFORD, UK

organised by:



VECTOR CONTROLLED INDUCTION MOTOR DRIVE MODELLING USING VHDL.

A. Aounis, M. Cirstea, M. McCormick, A. Dinu,

Electronic and Electrical Engineering Department,
De Montfort University, Leicester, UK
E. mail: Aounis@yahoo.com

Abstract: The paper presents a new approach to the modelling, simulation and controller design of a complete induction motor electric drive system. The novel technique uses a hardware description language (VHDL) as a unique EDA environment for the system modelling, evaluation and controller design. Simulation results are presented, proving the validity of the model for the vector controlled induction motor system. The advantages of the VHDL approach for a complete drive modelling and control strategy implementation are underlined. Copyright 2000 © IFAC

Keywords: induction motor, vector control, VHDL modelling

1. INTRODUCTION

Induction motors are perhaps the most rugged and best-understood motors presently available. It has been estimated that they are used in seventy to eighty percent of all industrial drive applications, the majority being fixed-speed applications such as pump or fan drives. The main advantages of induction motors are their simple maintenance and cost effective operation. Compared with brushed motors, a.c. motors can be designed to give substantially higher output ratings with lower weights and lower inertia, and they do not have the problems associated with the maintenance of commutators and brush gears (Gottlieb, 1994; Crowder, 1995).

Various current control techniques for Pulse Width Modulated (PWM) inverters have been investigated and reported in the literature (e.g. Katsunari, *et al.*, 1998; David, and Donald, 1985; Nabae, 1986).

The application of vector control techniques in a.c. drives demands accurate position and speed feedback information for the current control and servo-control loops. In many applications, the rotor position signal is obtained from a mechanical sensor, such as an optical encoder or a resolver, that may reduce the system's reliability and add significantly to the drive costs. Consequently, a strong interest arises in the alternative sensorless control, using only stator voltage and current measurement based on state observers. State observers are usually implemented to reconstruct the inaccessible states of the controlled process. They are especially useful for full-state feedback control, developed on state-space theory, in that the combined observer-controller system can easily be designed to meet specific qualitative and quantitative requirements. Several control algorithms were developed in the late 1960's by which a.c. induction motors could nearly achieve torque control.

However, these methods met with only limited success, since they required too many and complicated calculations that involved many unknown model parameters and numerous approximations. With the availability of fast switching and more easily controlled power electronic components such as the Gate Turn Off thyristor (GTO), the Bipolar Junction Transistor (BJT) and the IGBT's, research engineers realised that new control schemes could be implemented for a.c. machines.

In the 1970's a new algorithm for a.c. induction motor control, known as Field Oriented Control (FOC) was introduced. It applied a two- vector method to the induction motor by separating the stator current into a flux-producing component and an orthogonal torque-producing component. Therefore, field orientation provides the same de-coupled control of torque and flux as with the d.c. machine.

2. SYSTEM MODELLING

In order to obtain the performance required by servo applications, induction motors control is achieved using the vector control strategy. This allows high performance control of torque, speed or position to be achieved from an induction machine (Katsunari, *et al.*, 1998). The method can provide at least the same performance from an inverter-driven induction motor as it is obtainable from a separately excited d.c. machine. Vector control provides de-coupled control of rotor flux magnitude and the current component generating the torque.

The development approach adopted for the system presented in this paper combines the vector control strategy with a new modelling and design approach. The complete drive system was modelled, simulated and evaluated using VHDL. Very High Speed Integrated Circuit Hardware Description Language (VHDL) is now one of the most popular standard HDLs, comprehensively described by (Perry, 1998; Navabi, 1998). It is supported by all major Computer Aided Engineering (CAE) platforms and synthesis tools can compile VHDL designs into a large variety of target technologies.

The complete system was modelled and simulated in VHDL and then the digital controller was designed using VHDL. This will be synthesised and downloaded into a XILINX FPGA for rapid prototyping. The VHDL approach presented in this paper provides important advantages such as: wide compatibility of the design with respect to different CAE software tools (VHDL files are ASCII files), a large range of implementation technologies and the reuse of the VHDL code.

3.THE FOC ALGORITHM STRUCTURE

High performance control of a.c. induction motors and permanent magnet synchronous motors most often relies on the principles of vector or field oriented control. Vector controllers mainly aim to maintain the flux producing the direct component of the stator current space vector in phase with the rotor flux space vector under all operating conditions. The quadrature axis current components, which then lies in quadrature with the rotor flux vector, directly controls the torque developed by the machine. When correctly implemented, vector control permits the independent control of the torque and the flux of the a.c. machines in a manner identical to the separately excited d.c. motor.

Many different vector control structures are possible for a.c. induction motor depending on the desired performance level and the acceptable implementation. Both direct and indirect vector controller structures are possible, depending on whether or not there is a direct measurement or estimation of the flux quantity, to which the current must be oriented.

Most often there is no direct measurement of either the produced torque or flux so that the control is implemented by a closed loop current regulation whose references are derived from a feedforward control structure for the induction motor known as the Indirect Rotor Field Oriented Controller. Such a system is illustrated in Figure1., (Tazi and Monmasson, 1998; Krafka and Bunt, 1995).

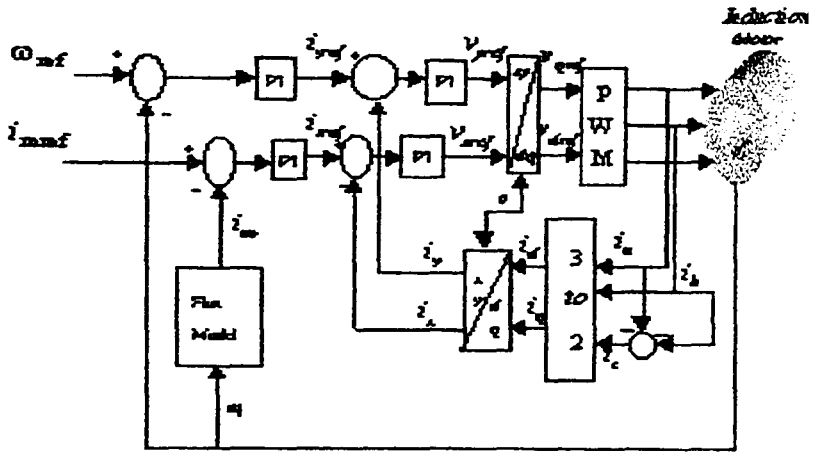


Fig. 1. Control principal of a.c. machine

The Induction Machine is supplied by a voltage-source PWM inverter. The output voltages of the inverter are controlled by a pulse width modulation technique.

The flux model shown in Figure 2 generates the angle ρ_r, which is used in the transformation blocks. Furthermore, the flux model is used to obtain the angular speed of the rotor flux ω_{mr} and the modulus of the magnetising current $|\bar{i}_{mr}|$, since these are also used in the decoupling circuit. The modulus of the rotor magnetising current is also used to obtain the electromagnetic torque. Vector control works on the principle of measuring two phase currents i_a, i_b and then the third one i_c is calculated from $i_c = -(i_a + i_b)$ and are transformed into current components in the (d,q) rotating frame.

The speed controller, of type PI, provides the reference torque (t_{eref}). The torque controller, again of PI type, gives the reference value of the quadrature axis stator current in the rotor flux oriented reference frame (i_{syref}). The reference signal $|i_{mref}|$ is compared with the actual value of the rotor magnetising current and the error serves as input to the flux controller which is a PI controller. Its output is the direct axis stator current reference (i_{sxref}). The error signals ($i_{sxref} - i_{sx}$) and ($i_{syref} - i_{sy}$) are the inputs to the respective current controllers. The outputs of these

controllers are added to the corresponding outputs of the decoupling circuits. Thus, the direct and the quadrature axis reference stator voltages v_{xref} and v_{yref} are obtained and therefore they have to be transformed by $e^{j\rho_r}$ to obtain the two axis reference stator voltages in the stationary reference frame (v_{qref}, v_{dref}). This is followed by the 2-phase to 3-phase transformation.

4. THE FLUX MODEL

The modulus and phase angle of the rotor flux phasor must be calculated to obtain ω_{mr} and $|\bar{i}_{mr}|$. The space phasor of the rotor magnetising currents expressed in the magnetising flux oriented reference frame (Vas, 1990) can be found from the following equation:

$$T_r \frac{d|\bar{i}_{mr}|}{dt} + |\bar{i}_{mr}| = i_{sx} \qquad (1)$$

$$\omega_{mr} = \omega_r + \frac{i_{sy}}{T_r |\bar{i}_{mr}|} \quad \ldots \qquad \ldots.(2)$$

Based on the above equations a rotor flux can be determined, as shown by the block diagram in Figure 2. The angular slip frequency of the rotor flux is:

$$\omega_{sl} = \frac{i_{sy}}{T_r |\bar{i}_{mr}|} \quad \ldots\ldots\ldots\ldots\ldots \quad .(3)$$

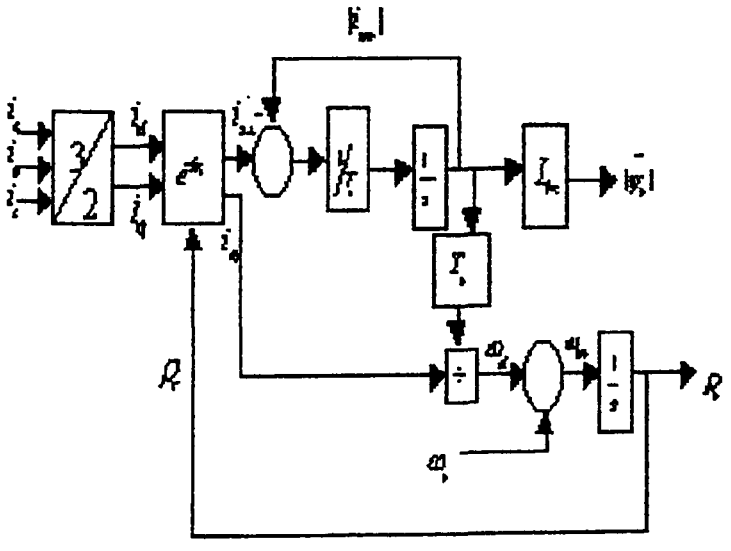


Fig 2: Flux model in the rotor reference frame.

5. SIMULATION RESULTS

The analysis and simulation of the control algorithm was achieved using behavioral VHDL programs (appendix A). Figure 3 shows the waveform of the stator voltage versus time. Figures 4 and 5 illustrate the time response of the rotating speed and the electromagnetic torque, when a start from zero speed is performed and at no load torque. During the transient state, the electromagnetic torque increases to its maximum value and once the speed reaches its reference the torque drops to approximately zero. Figure 6 shows a good tracking of the actual speed to the reference speed while figure 7 shows an expected torque and sector response.

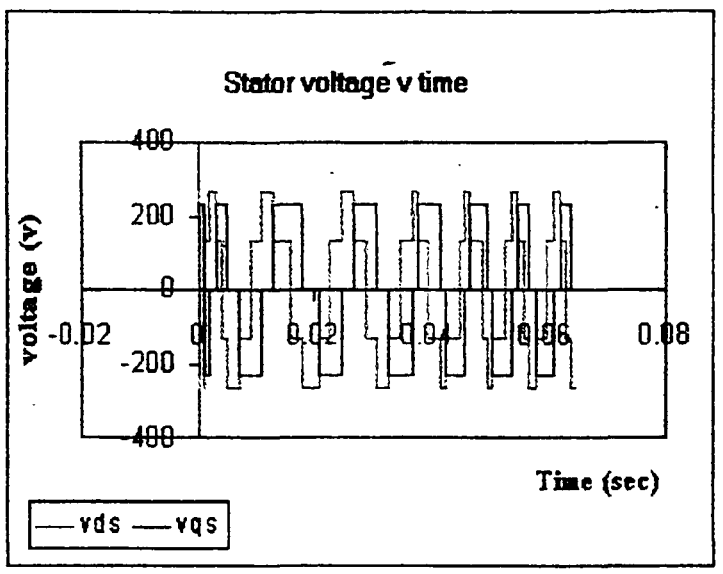


Fig. 3. Stator voltage versus time.

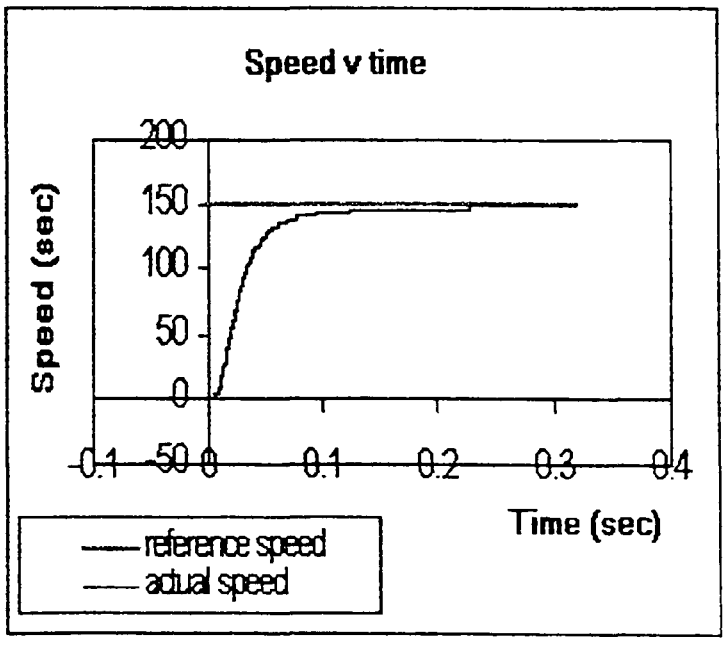


Fig. 4. Speed and its reference.

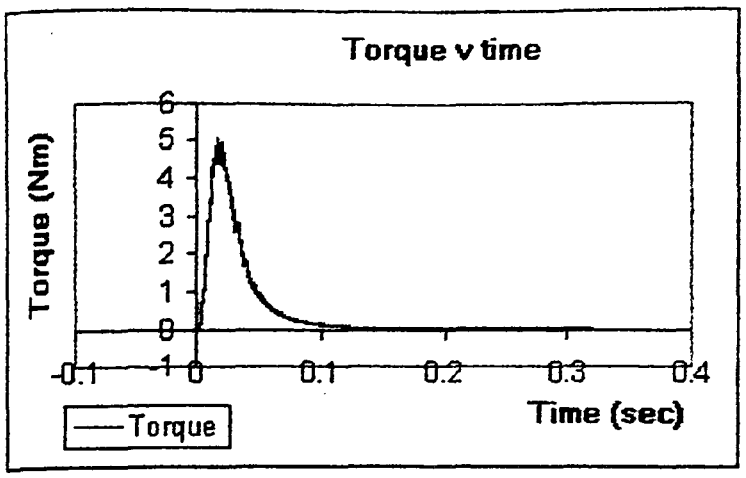


Fig. 5. Torque versus time

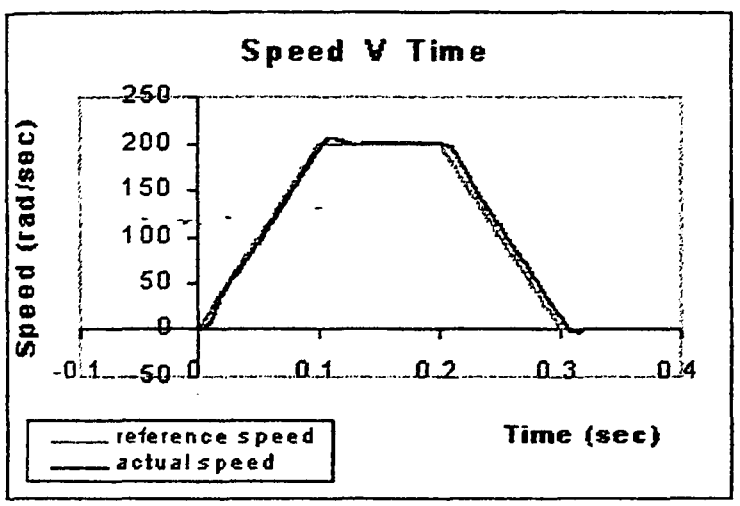


Fig. 6. Reference and actual rotor speed

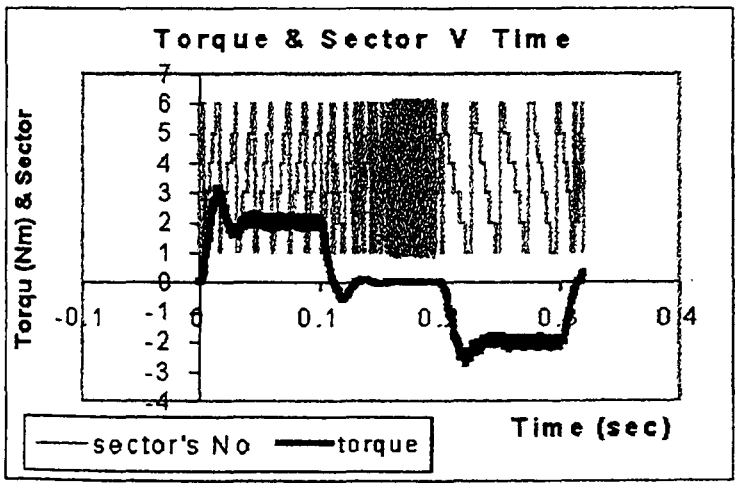


Fig. 7. Torque and sector versus time

6. CONCLUSIONS

A new approach has been developed for the modeling and design of a complete vector controlled induction motor drive system. The simulation results of the VHDL model have been presented, proving an expected behavior of the motor model. There are major advantages of the new approach, such as:

- A unique environment for modeling, simulation and evaluation of complete drive systems, including controllers, power electronics and induction motors.
- The same environment (VHDL) is used for the design itself of the digital controller achieving the vector control strategy and for silicon (FPGA) implementation.
- Fast design development and short time to market.
- A CAD platform independent model and design are being developed (VHDL operates with ASCII files) and therefore a valuable IP can be produced, in co-relation with the modern principles of design reuse.
- Concurrent engineering basic rules (unique EDA environment and common design database) are fulfilled.
- The digital controller circuit design is currently being synthesized. XILINX FPGA implementation is targeted.
- The extensive use of VHDL for drive systems modeling and simulation, in conjunction with accurate digital controllers design is foreseen for the near future.

REFERENCES

Crowder, R. M. (1995). *Electric Drives and their Controls* (Oxford University Press Inc).

David, M. and W. Donald (1985). *Current Control of VSI-PWM Inverters*, IEEE Trans. on power electronics, **vol. IA-21, No 4**, pp. 562-570.

Gottlieb, M. I. (1994). *Electric Motors and Control Techniques* (McGraw Hill).

Katsunori, T., O. Yasumasa and I. Hisaichi (1998). *PWM Technique For Power MOSFET Inverter*, IEEE Trans. on power electronics, **Vol.3, No.3**, pp.328-334.

Krafka, P. and A. Bunte and H. Grotstollen. (1995). *Novel Control Scheme for Current controlled PWM Inverters*, IEEE Trans.on power electronics, **vol. IA-22, No 4**, pp. 697-701.

Nabae, A., S. Ogasaward and H. Akagi (1986). *A Novel Control Scheme for Current Controlled PWM Inverters*, IEEE Trans. on power electronics, **vol. IA-22, No 4**, pp. 697-701.

Navabi, Z. (1998). *VHDL: Analysis and Modeling of Digital Systems*, (McGraw Hill).

Perry, D. (1998). *VHDL*, (McGraw-Hill).

Tazi, K. and E. Monmasson. (1998). *Single-Chip DSP Based Speed Control Of Two AC-Machine*, **Speedam, Sorrento (Italy)**,3-5, pp. P4-33-38.

Vas, P. (1990). *Vector Control of AC Machine* (ClarendonPress,Oxford, UK)

APPENDIX A

VHDL Motor Model File

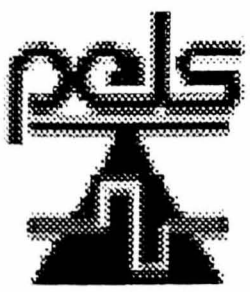
```
LIBRARY math;
USE math.mathtyx.all;
USE std.textio.all;
-- Electrical+mechanical model
ENTITY motor IS
 PORT (vds,vqs,Tl: IN Real;
    ids,iqs,wr: OUT REAL);
END motor;
ARCHITECTURE arch_motor OF motor IS
 CONSTANT Rs: REAL := 5.9;
 CONSTANT Rr: REAL := 4.62;
 CONSTANT ls: REAL := 0.831;
 CONSTANT lr: REAL := 0.833;
 CONSTANT lm: REAL := 0.809;
 CONSTANT jr: REAL := 0.001;
 CONSTANT deltat:TIME:= 1000ns;
 CONSTANT dt: REAL :=1.0e-6;
 CONSTANT wc: REAL := 50.0;
 CONSTANT p: REAL := 4.0;
 CONSTANT FG: REAL := 0.05;
 constant flag:integer:=1;
-- *** Speed controller ***
 CONSTANT ki: REAL :=0.05;
 CONSTANT kp: REAL :=1;
-- *** Torque Controller ***
 CONSTANT kit: REAL :=0.01;
 CONSTANT kpt: REAL :=1;
-- *** iy Current controller ***
 CONSTANT kci: REAL :=3;
 CONSTANT kcp: REAL :=100;
-- *** ix current controller ***
 CONSTANT kixi: REAL :=3.8;
 CONSTANT kixp: REAL :=100.5;
-- *** Flux controller ***
 CONSTANT kmci: REAL :=3.0;
 CONSTANT kmcp: REAL :=100.0;
CONSTANT imref: REAL :=0.005;
signal next_step: INTEGER := 1;
signal vdsr,vqsr: REAL :=0.0;
signal vdsr1,vqsr1: REAL :=0.0;
signal tt:real:=1.0;
FILE outf: TEXT IS OUT "C:\motor.txt";
BEGIN
PROCESS(next_step)
VARIABLE my_line: LINE;
VARIABLE a,ids1,idss,iqs1,iqss: REAL:=0.0;
VARIABLE idr1,idrr,iqr1,iqrr: REAL:=0.0;
VARIABLE Te,wr1,wrr: REAL :=0.0;
variable tetar,tr,imr,imr1,isx,fluxr,
      isy,wmr,tetamr :real:=0.0;
VARIABLE dif,difi,Trq,ert,teref,erti,
      isyref: REAL :=0.0;
VARIABLE cer,ceri,lss,vdx,vq,vsy,vdy:
      REAL :=0.0;
VARIABLE mce,mcei,isxref: REAL :=0.0;
VARIABLE cxe,cxei,vsx,vd: REAL :=0.0;
variable t:real:=0.0;
CONSTANT d_space: STRING :="  ";
BEGIN
 IF next_step=1 THEN
   WRITE(my_line,wc);
   WRITE (my_line,d_space);
   WRITE(my_line,wrr);
   WRITE (my_line,d_space);
   WRITE(my_line,vdsr);
   WRITE (my_line,d_space);
   WRITE(my_line,vds);
   WRITE (my_line,d_space);
   WRITE(my_line,tt);
   WRITE (my_line,d_space);
   WRITE(my_line,t);
   WRITE (my_line,d_space);
   WRITELINE(outf,my_line);
 END IF;
  Tr:=rr/lr;
  a:=(lm*lm-lr*ls);
  IF tt =1.0 then
     vdsr1 <= vds;
     vqsr1 <= vqs;
     tt <= 0.0;
    else
     vdsr1 <= vdsr;
     vqsr1 <= vqsr;
   END IF;
ids1:=(rs*lr*idss-wrr*lm*lm*iqss-rr*
  lm*idrr-wrr*lr*lm*iqrr-lr*vdsr1)/a;
iqs1:=(wrr*lm*lm*idss+rs*lr*iqss+wrr*
   lr*lm*idrr-rr*lm*iqrr-lr*vqsr1)/a;
idr1:=-(rs*lm*idss-wrr*lm*ls*iqss-rr*
  ls*idrr-wrr*lr*ls*iqrr-lm*vdsr1)/a;
iqr1:=-(wrr*lm*ls*idss+rs*lm*iqss+
wrr*lr*ls*idrr-rr*ls*iqrr-lm*vqsr1)/a;
idss:=idss+(ids1*dt);
iqss:=iqss+iqs1*dt;
idrr:=idrr+idr1*dt;
iqrr:=iqrr+iqr1*dt;
Te:=p*(3.0/2.0)*lm*(iqss*idrr-idss*
    iqrr);
wr1:=(Te-Tl)/jr;
wrr:=wrr+wr1*dt;
tetar:=tetar+wrr*dt;
t:=t+dt;
-- End of electrical+mechanical model
-- *** Vector Control ***
   imr1:=(isx-imr)/Tr;
   imr:=imr+imr1*dt;
   fluxr:=lm*imr;
   if imr >0.0 then
      wmr:= wrr+(isy/(Tr*imr));
   end if;
   tetamr:=tetamr+wmr*dt;
```

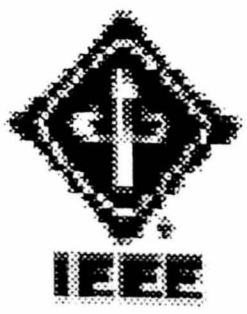
```
    isx:=idss*cos(tetamr)+
        iqss*sin(tetamr);
    isy:=-idss*sin(tetamr)+
         iqss*cos(tetamr);
-- *** Speed control loop ***
    dif :=(wc-wrr);
    difi:=difi+dif*dt;
    teref:=(ki*difi+kp*dif);
-- *** Actual Torque  ***
    Trq := p*(3.0/2.0)*
         (lm*lm/lr)*imr*isy;

-- *** Torque control ***
    ert:=teref-trq;
    erti:=erti+ert*dt;
    isyref:=(kit*erti+kpt*ert);
-- *** iy current control ***
    cer:= isyref-isy;
    ceri:=ceri+cer*dt;
    vsy:=(kci*ceri+kcp*cer);
-- *** Decoupling block ***
lss:=ls-(lm*lm/lr);
vdy:=wmr*lss*isx+(ls-lss)*wmr*imr;
vdx:=-wmr*lss*isy;
vq:=vsy+vdy;

-- *** Flux controller ***
    mce:=imref-imr;
END conf_motor;
```

```
    mcei:=mcei+mce*dt;
    isxref:=(kmci*mcei+kmcp*mce);
-- *** ix current controller  ***
    cxe:=isxref-isx;
    cxei:=cxei+cxe*dt;
    vsx:=(kixi*cxei+kixp*cxe);
    vd:=vsx+vdx;

-- ********************************
vdsr <= vd*cos(tetamr)-vq*sin(tetamr);
vqsr <= vd*sin(tetamr)+vq*cos(tetamr);
-- ********************************

IF next_step<500 THEN
next_step<=next_step+1 AFTER deltat;
ELSE
next_step<=1 AFTER deltat;
END IF;
ids<=idss;
iqs<=iqss;
wr<=wrr;
END PROCESS;
END arch_motor;

CONFIGURATION conf_motor OF motor IS
 FOR arch_motor
 END FOR;
```

32nd Power Electronics Specialists Conference

The University of British Columbia
Vancouver, Canada
June 17-22, 2001





Vector Control System Design and Analysis Using VHDL

M. Cirstea, A. Aounis, M. McCormick, P. Urwin
De Montfort University,
The Gateway, Leicester, LE1 9BH, UK

Abstract - **The paper is concerned with a new approach to the modelling, simulation, design and implementation of a vector control strategy for induction motors. The novel technique uses a hardware description language (VHDL) as unique EDA environment for all phases of the design process. Easy FPGA prototyping is facilitated and the modular design allows the reuse of VHDL code for a range of vector control strategies. Simulation results are presented, validating the vector control scheme model.**

I. Introduction

Induction motors are perhaps the most rugged and best understood motors presently available. Due to the complexity of the equations describing their behaviour, the control systems are complicated and expensive. On the other hand, it has been estimated that induction motors are used in 70-80% of all industrial drive applications due to their simple mechanical construction, low maintenance requirement and lower cost compared to brushless d.c. motors.

General research in the area of induction motors control is characterised by a large variety of control methodologies [1], [2], [3]. Although many a.c. drives in use today adopt microprocessor/DSP based digital control [4], the implementation of current loop and PWM control are still tied to analogue control circuitry. This kind of control scheme has the advantages of fast dynamic response, but suffers the disadvantage of complex circuitry, limited functions and difficulty in circuit modification [5]. Recently, motor control employing ASICs/FPGAs is receiving increased attention.

This paper presents a study of modelling, design and simulation of a reusable digital architecture for induction motor vector control, using VHDL and targeting FPGA implementation.

II. System Modelling

In order to obtain the performance required by servo applications, induction motor control is achieved using the vector control strategy. This allows high performance control of torque, speed or position to be achieved. The complete drive system was modelled, simulated and evaluated using Very High Speed Integrated Circuit Hardware Description Language (VHDL). This is now one of the most popular standard HDLs, comprehensively described in [6]. It is supported by all major Computer Aided Engineering (CAE) platforms and synthesis tools can compile VHDL designs into a large variety of target technologies. The approach presented in this paper provides important advantages such as: wide compatibility of the design with respect to different CAE software tools, a large range of implementation technologies and the reuse of the VHDL code. FPGAs are implemented with regular flexible programmable architecture of Configurable Logic Blocks (CLBs) and have generous routing resources to accommodate the most complex interconnected pattern.

FPGAs are customised by loading configuration data into internal static memory cells. Reprogramming is possible an unlimited number of times. FPGAs are therefore ideal for shortening design and development cycles and offering a cost-effective solution. The VHDL digital control solution presented in this paper is reusable as a whole or parts of it in different vector control architectures for induction motor control.

III. Digital Controller VHDL Design

High performance control of a.c. induction motors and permanent magnet synchronous motors most often relies on the principles of vector or Field Oriented Control (FOC). Vector controllers mainly aim to maintain the flux producing the direct component of the stator current space vector in phase with the rotor flux space vector under all operating conditions. The quadrature axis current component, which then lies in quadrature with the rotor flux vector, directly controls the torque developed by the machine.

When correctly implemented, vector control permits the independent control of the torque and flux of the a.c. machines, in a manner identical to that of the separately excited d.c. motor. Most often there is no direct measurement of either the produced torque or flux, so the control is implemented by a closed loop current regulation structure known as the Indirect Rotor Field Oriented Controller [4]. Such a system is illustrated in Fig.1.

Although VHDL is a hardware description language and as such, it is used primarily for circuit design, it has the basic properties of any software programming language. A system model can therefore be initially developed.

A vector control structure with the entity named "motor" can be configured in VHDL as:

```
ENTITY MOTOR IS
        PORT (vds,vqs,Tl :in real:
                ids, iqs,wr : out real);
END MOTOR;
```

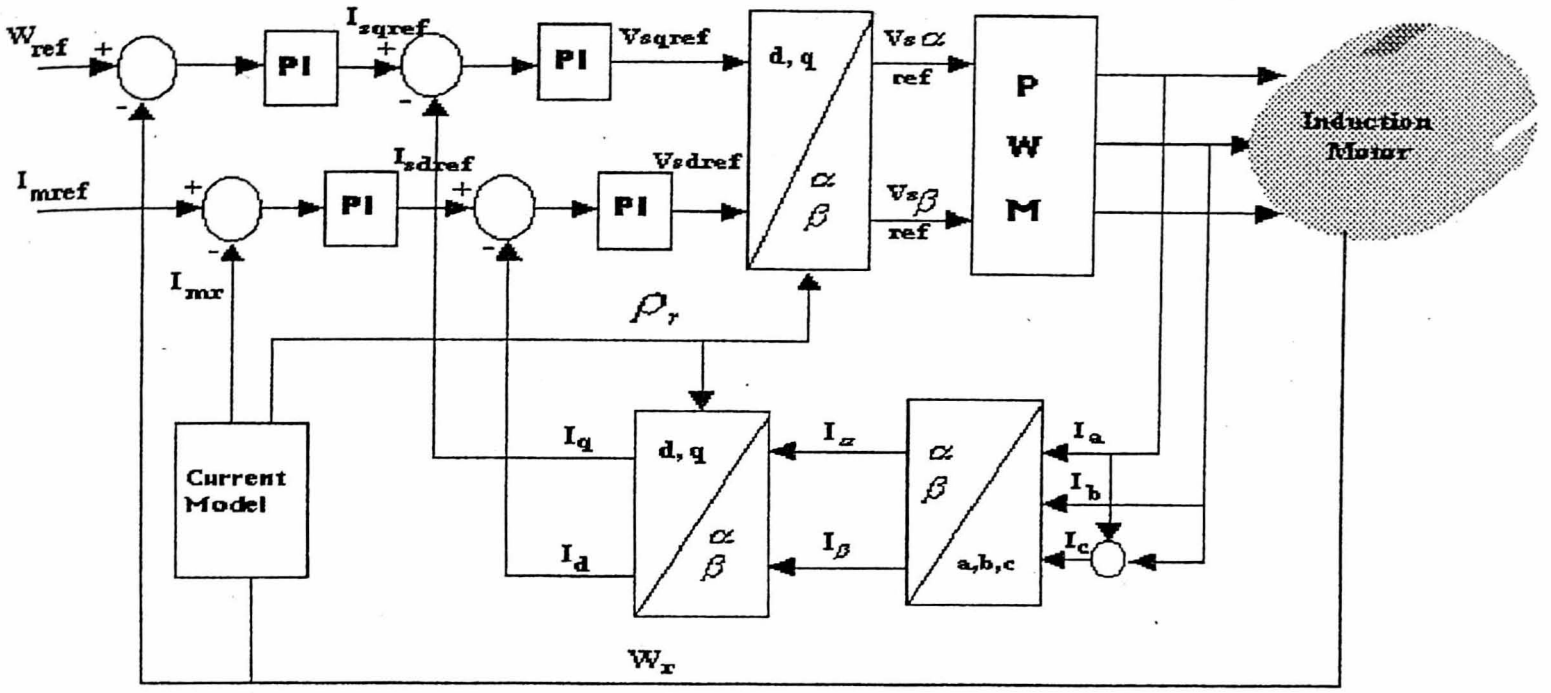


Fig.1 Vector control principle of a.c. machine

The input signals of the model are voltages in the d-q axis as well as the load torque, while the currents in the d-q axis are the output signals. The differential equations of the a.c. motor and the control algorithm are described in the architecture. To verify the mathematical model, a VHDL functional simulation is performed, based on Euler's integration method. The results illustrated in Fig.2 show good tracking of the reference speed by the actual motor speed.

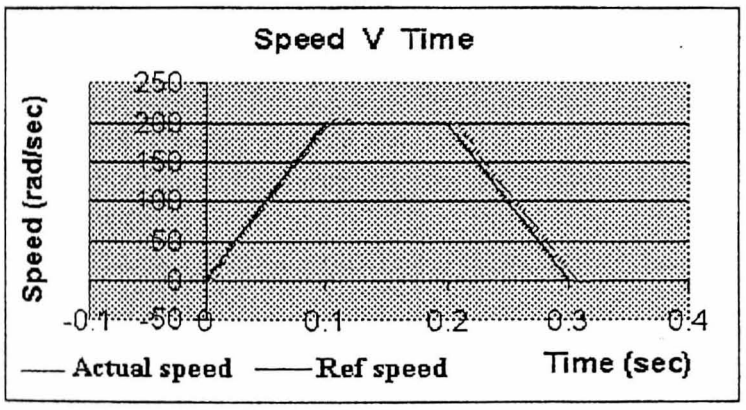


Fig.2 Simulated speed profile of the motor model

IV. FPGA DESIGN FOR IMPLEMENTATION

So far the design of FOC did not take into account the synthesis and implementation considerations and the limitations of the target technology. Before VHDL design can be synthesized, all the VHDL functions which are not synthesisable (e.g. declaring a REAL number) have to be eliminated and replaced with functions that can be translated into hardware. The controller is split into components, each performing a specific function. Two of these are discussed.

A. *Clark Transform*

The Clark transform (Fig.3) is based on the principle of using an intermediate coordinate system α, β and its phase current projections i_α and i_β. Therefore:

$$i_\alpha = i_a \tag{1}$$

$$i_\beta = \frac{1}{\sqrt{3}}(2 * i_b + i_a) \tag{2}$$

The ADC samples the currents (i_a and i_b) and converts them into a nine-bit signed binary number. Equation 1 is easily implemented in VHDL, while equation 2 is achieved as purely combinational circuit using a single PROCESS statement. Entity "Clarktransform" is achieved by specifying the input signals (i_a, i_b), output signals (i_{alpha}, i_{Beta}), and the data types:

```
Entity Clarktransform is
  Port (
    ia,ib:IN Std_Logic_Vector (8 downto 0);
    ialpha,ibeta: OUT Std_Logic_Vector (9 downto 0);
End Clarktransform;
```

The functionality is described in the architecture body:

```
begin
ialpha<=(ia(8) & ia);
ibeta_par<=(ib & '0') + ia;
process(ibeta_par)
variable out: std_logic_vector(21 downto 0);
begin
    out := ibeta_par & zeroes(12);
    out := out - (ibeta_par & zeroes(10));
    out := out - (ibeta_par & zeroes(9));
    out := out - (ibeta_par & zeroes(7));
    out := out - (ibeta_par & zeroes(6));
    out := out - (ibeta_par & zeroes(1));
    out := out - ibeta_par;
  ibeta<=out(21 downto 12);
end process;
```

The VHDL description was simulated (Fig.4) to confirm the correct behaviour.

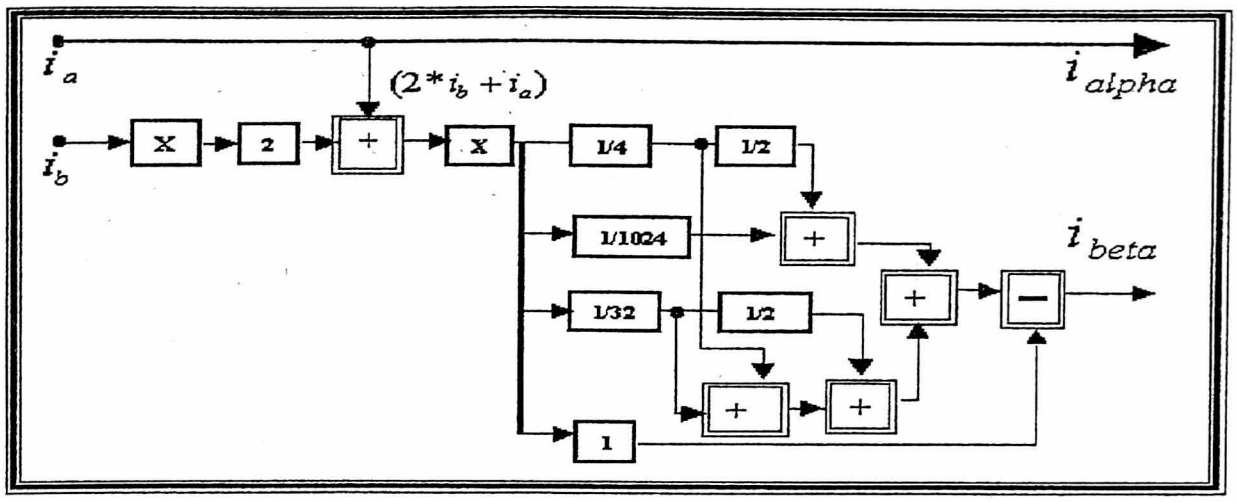


Fig.3 Circuit of the 3phase- 2 axis current converter (Clark Transform)

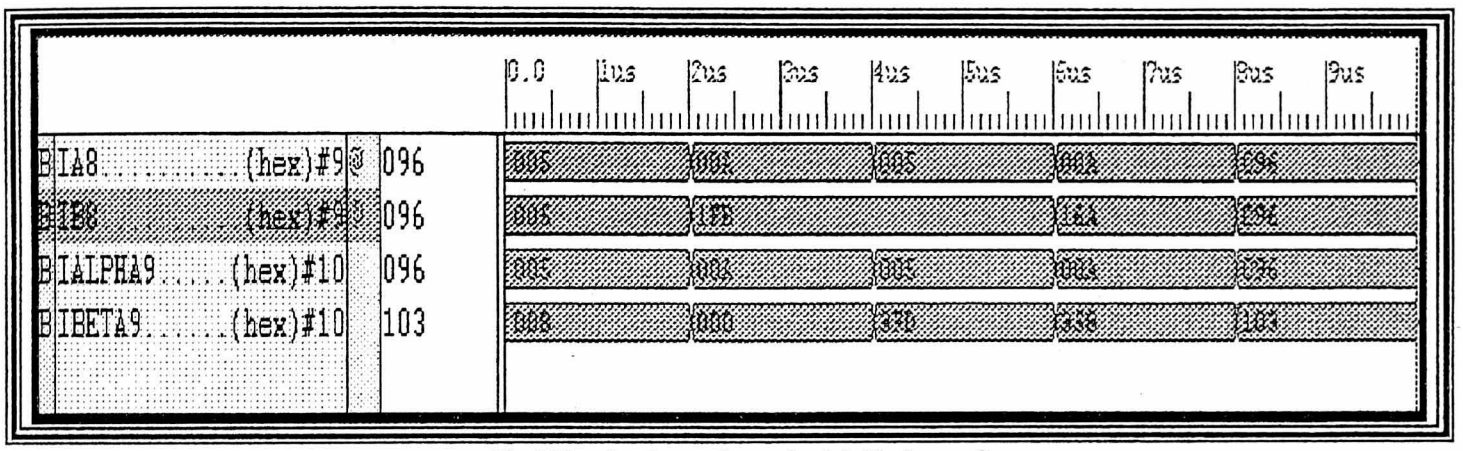


Fig.4 Simulated waveforms for 3-2 Clark transform

B. Park Transform

Park transformation is based on modifying the two phase orthogonal system (α, β) into the (d, q) rotating reference frame [7]. The relation between the two reference frames can be expressed as:

$$i_{sd} = i_{alpha} * \cos(teta) + i_{beta} * \sin(teta) \quad (3)$$

$$i_{sq} = -i_{alpha} * \sin(teta) + i_{beta} * \cos(teta) \quad (4)$$

The equations are represented in Fig.5 as a block diagram. The VHDL structure of Park transform consists of: control unit, multiplier, registers (1 to 4), adder, subtractor. Simulation results are illustrated in Fig.6.

The two blocks presented in this section were then integrated with the others (Fig.1), developed in a similar manner. The complete controller design was implemented in a single Xilinx FPGA (Spartan S40PQ208). Hardware tests are currently being performed, the first set of experiments showing a very good match of the simulation results.

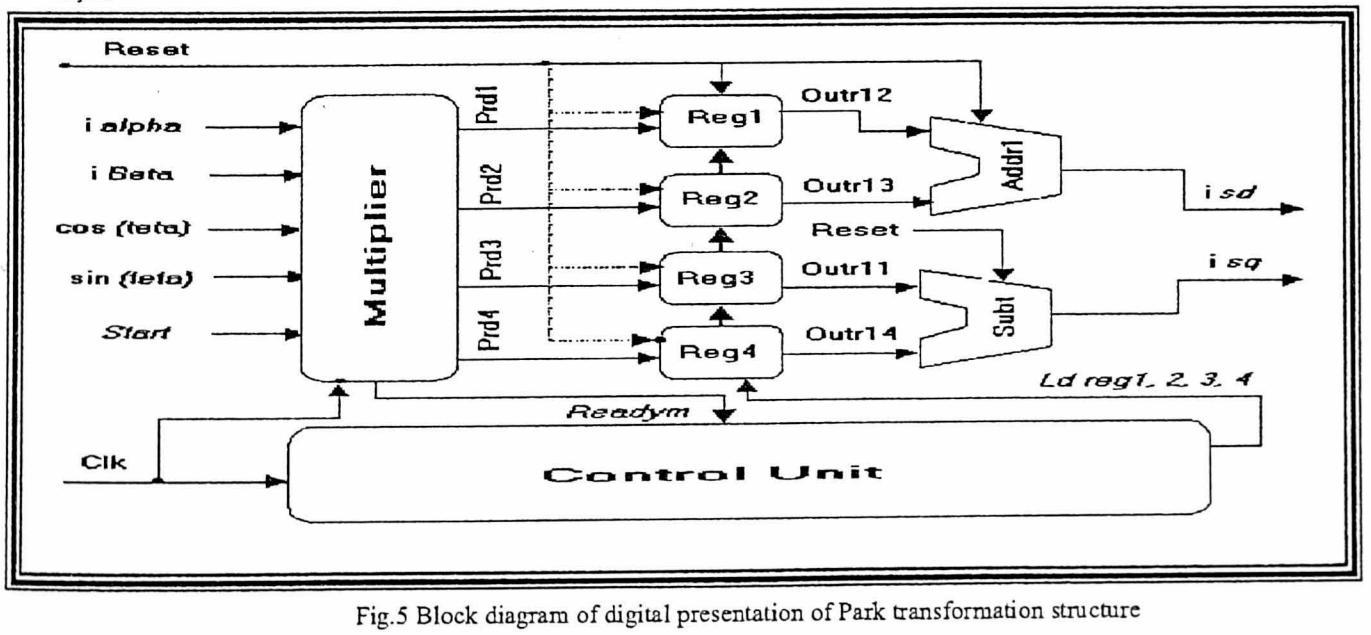


Fig.5 Block diagram of digital presentation of Park transformation structure

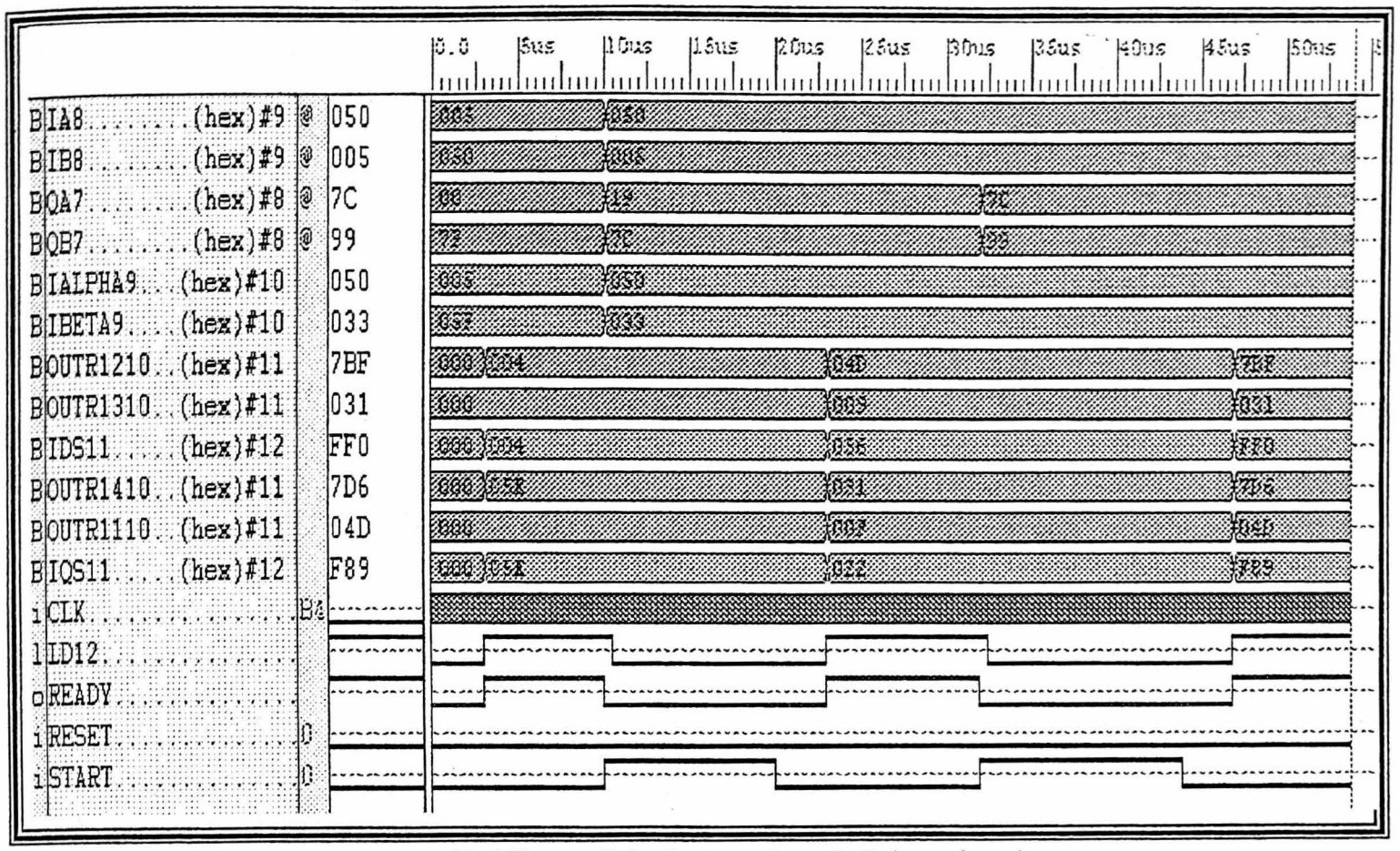


Fig.6 The overall simulated waveforms for Park transformation

V. CONCLUSIONS

A new approach has been developed for the modelling, design and analysis of a complete vector controlled induction motor drive. Two reusable VHDL modules are presented, together with simulation results. These prove an expected behaviour of the motor model.

Some important advantages of the method are:

- A unique environment for modelling, simulation and evaluation of complete drive systems, including controllers, power electronics and induction motors.
- The same environment (VHDL) is used for the design itself of the digital vector controller and for silicon (FPGA) implementation.
- Fast design development and short time to market.
- CAD platform independent models and designs are being developed (VHDL operates with ASCII files) and therefore valuable reusable IPs can be produced.

The further development of holistic modelling methods based on hardware description languages for power systems analysis is envisaged by the authors.

REFERENCES

[1] T. Katsunori, O. Yasumasa and I. Hisaichi, "PWM Technique For Power MOSFET Inverter", *IEEE Trans. on power electronics*, Vol.3, No.3, pp.328-334, 1998.

[2] M. David and W. Donald, "Current Control of VSI-PWM Inverters", *IEEE Trans. on power electronics*, vol. IA-21, No 4, pp. 562-570, May/June. 1985.

[3] A. Nabae, S. Ogasaward and H. Akagi, "A Novel Control Scheme for Current Controlled PWM Inverters", *IEEE Trans. on power electronics*, vol.IA-22, No 4, pp.697-701, July/August 1986.

[4] K. Tazi and E. Monmasson, "Single-chip DSP based speed control of two a.c. machines", *SPEEDAM Machines and Drives Conference*, Italy, pp.P4-33-P4-38, June 1998.

[5] Y. Tzou and H. Jean, "FPGA realization of space vector PWM control IC for three phase PWM inverters", *IEEE Trans. on power electronics*, vol. 22, No 6, pp.953-963, November 1997.

[6] D. Perry, "VHDL", McGraw-Hill, Second Edition, 1998.

[7] P. Vas, "Vector Control of AC Machines", Clarendon Press, Oxford, 1990.

Proceedings

VHDL International Users Forum Fall Workshop

Orlando, Florida

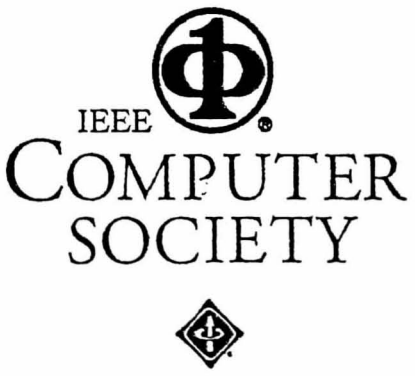
18-20 October 2000

Sponsored by

IEEE Computer Society Technical Committee on Design Automation (DATC)

IEEE Circuits and Systems Society

VHDL International



Los Alamitos, California

Washington • Brussels • Tokyo

The development approach adopted for the system presented in this paper combines the vector control strategy with a new modelling and design approach. The complete drive system was modelled, simulated and evaluated using VHDL. Very High Speed Integrated Circuit Hardware Description Language (VHDL) is now one of the most popular standard HDLs, comprehensively described by Perry [6] and Navabi [7]. It is supported by all major Computer Aided Engineering (CAE) platforms and synthesis tools can compile VHDL designs into a large variety of target technologies.

The complete system was modelled and simulated in VHDL and then the digital controller was designed using VHDL. This will be synthesised and downloaded into a XILINX FPGA for rapid prototyping. The VHDL approach presented in this paper provides important advantages such as: wide compatibility of the design with respect to different CAE software tools (VHDL files are ASCII files), a large range of implementation technologies and the reuse of the VHDL code.

3. The Field Oriented Control

High performance control of a.c. induction motors and permanent magnet synchronous motors most often relies on the principles of vector or field oriented control. Vector controllers mainly aim to maintain the flux producing the direct component of the stator current space vector in phase with the rotor flux space vector under all operating conditions. The quadrature axis current components, which then lies in quadrature with the rotor flux vector, directly controls the torque developed by the machine. When correctly implemented, vector control permits the independent control of the torque and the flux of the a.c. machines in a manner identical to the separately excited d.c. motor.

Many different vector control structures are possible for a.c. induction motor depending on the desired performance level and the acceptable implementation. Both direct and indirect vector controller structures are possible, depending on whether or not there is a direct measurement or estimation of the flux quantity, to which the current must be oriented.

Most often there is no direct measurement of either the produced torque or flux so that the control is implemented by a closed loop current regulation whose references are derived from a feedforward control structure for the induction motor [8], known as the Indirect Rotor Field Oriented Controller. Such a system is illustrated in Fig.1. The Induction Machine is supplied by a voltage-source PWM inverter. The output voltages of the inverter are controlled by a pulse width modulation technique.

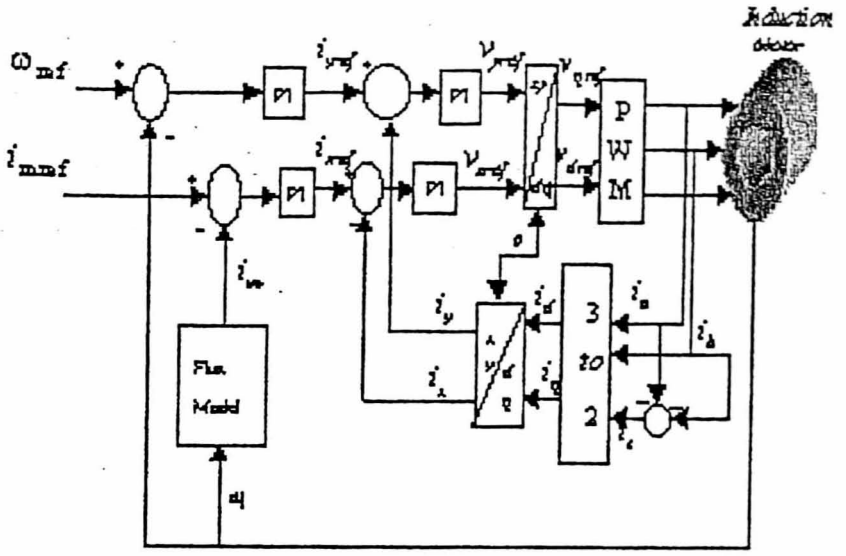


Fig. 1: Control principle scheme of a.c. machine.

The flux model shown in Fig. 2 generates the angle ρ_r, which is used in the transformation blocks. Furthermore, the flux model is used to obtain the angular speed of the rotor flux ω_{mr} and the modulus of the magnetizing current $|\bar{i}_{mr}|$, since these are also used in the decoupling circuit. The modulus of the rotor magnetizing current is also used to obtain the electromagnetic torque.
Vector control works on the principle of measuring two phase currents i_a, i_b and then the third one i_c is calculated from $i_c = -(i_a + i_b)$ and are transformed into current components in the (d,q) rotating frame.
The speed controller, of type PI, provides the reference torque (t_{eref}). The torque controller, again of PI type, gives the reference value of the quadrature axis stator current in the rotor flux oriented reference frame (i_{syref}). The reference signal $|i_{mref}|$ is compared with the actual value of the rotor magnetizing current and the error serves as input to the flux controller which is a PI controller. Its output is the direct axis stator current reference (i_{sxref}). The error signals ($i_{sxref} - i_{sx}$) and ($i_{syref} - i_{sy}$) are the inputs to the respective current controllers. The outputs of these controllers are added to the corresponding outputs of the decoupling circuits. Thus, the direct and the quadrature axis reference stator voltages v_{xref} and v_{yref} are obtained and therefore they have to be transformed by $e^{j\rho_r}$ to obtain the two axis reference stator voltages in the stationary reference frame (v_{qref}, v_{dref}). This is followed by the 2-phase to 3-phase transformation.

4. The Flux Model

The modulus and phase angle of the rotor flux phasor must be calculated to obtain ω_{mr} and $|\bar{i}_{mr}|$. The space phasor of the rotor magnetizing currents expressed in the magnetizing flux oriented reference frame [9] can be found from the following equation :

$$T_r \frac{d|\bar{i}_{mr}|}{dt} + |\bar{i}_{mr}| = i_{sx}$$

$$\omega_{mr} = \omega_r + \frac{i_{sy}}{T_r |\bar{i}_{mr}|}$$

Based on the above equations a rotor flux can be determined, as shown by the block diagram in Fig. 2. The angular slip frequency of the rotor flux is:

$$\omega_{sl} = \frac{i_{sy}}{T_r |\bar{i}_{mr}|}.$$

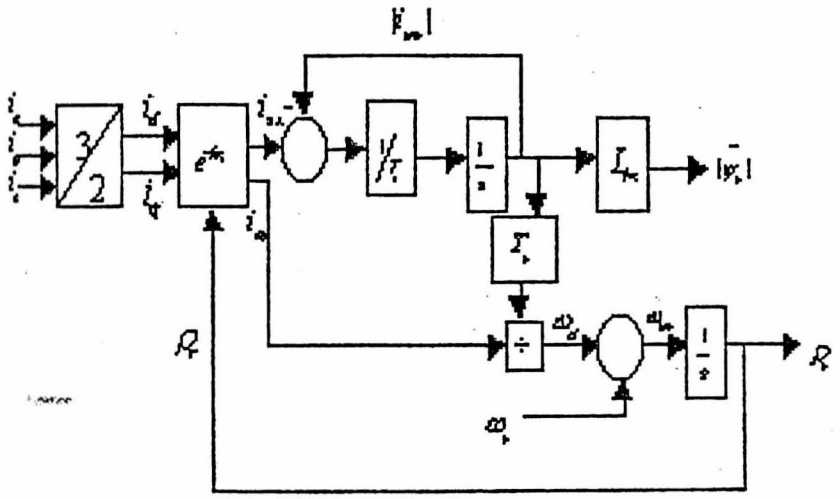


Fig. 2: Flux model in the rotor reference frame.

5. Simulation Results and Discussion

The analysis and simulation of the control algorithm was achieved using behavioral VHDL programs (appendix A). Figure 3 shows the waveform of the stator voltage in time. Figures 4, 5 and 6 illustrate the time response of the rotating speed and the electromagnetic torque, when a start from zero speed is performed and at no load torque.

During the transient state, the electromagnetic torque increases to its maximum value and once the speed reaches its reference the torque drops to approximately zero. Figure 6 shows a good tracking of the actual speed to the reference speed.

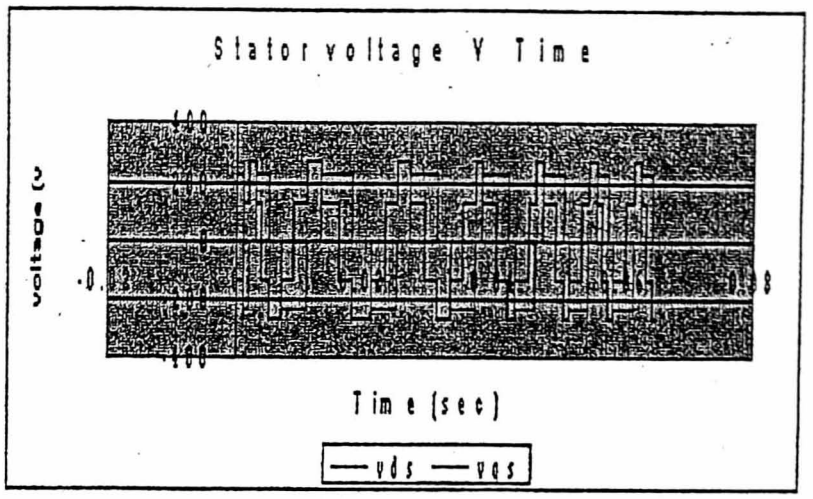


Fig. 3 Stator voltage versus time

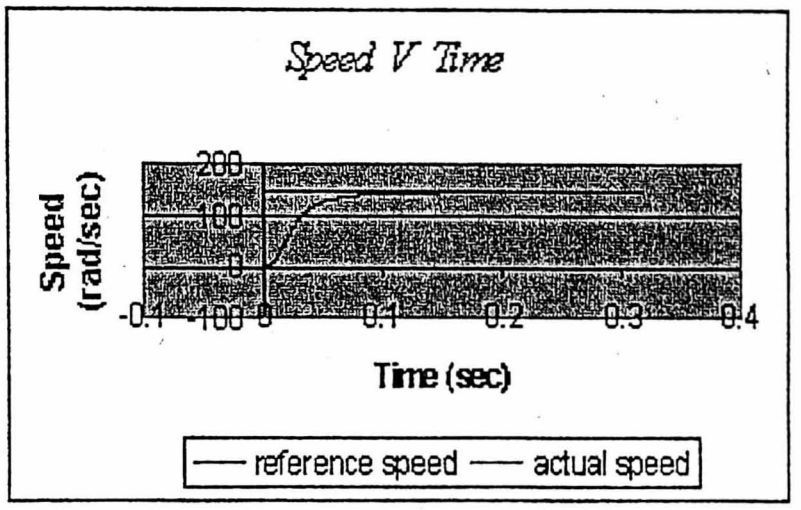


Fig.4 Speed and its reference

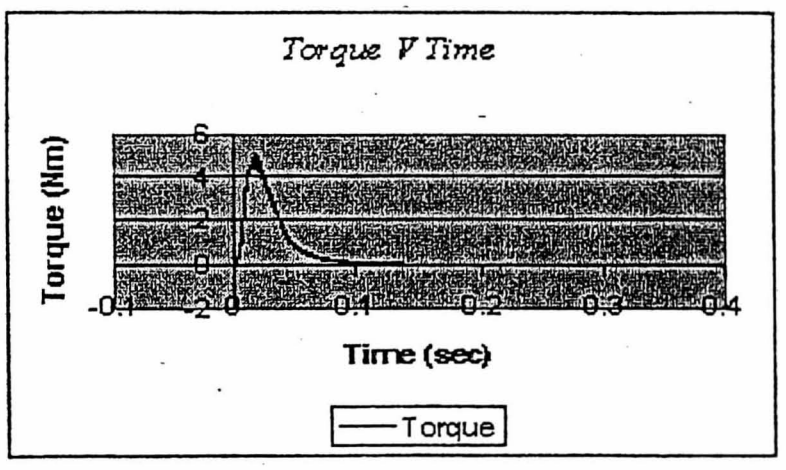


Fig. 5 Torque versus time.

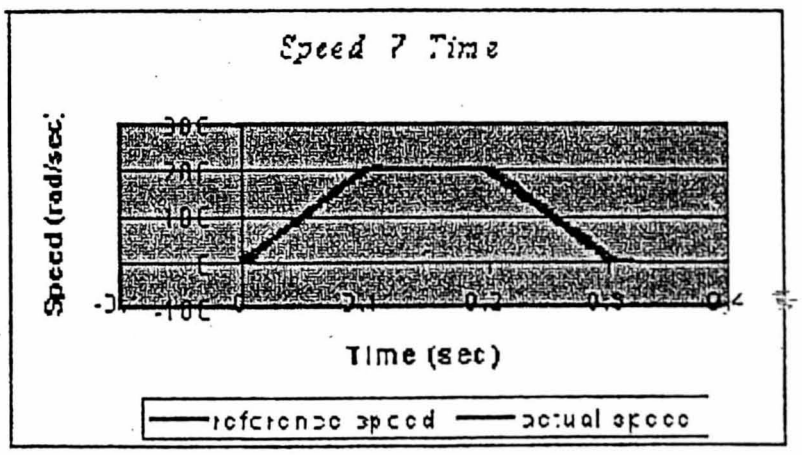


Fig. 6 Reference and actual rotor speed

6. Conclusions

At the time when this research work started VHDL-AMS tools were not available, therefore, an original VHDL approach was developed for the modelling and design of a vector controlled induction motor drive system. The simulation results are presented, proving an expected behaviour of the motor model. Important advantages can be identified, such as:

- A unique environment for modelling, simulation and evaluation of complete drive systems, including controllers, power electronics and induction motors.
- The same environment (VHDL) is used for the design itself of the digital controller achieving the vector control strategy and for silicon (FPGA) implementation.
- Fast design development and short time to market.
- A CAD platform independent model and design are being developed (VHDL operates with ASCII files) and therefore a valuable IP can be produced, in co-relation with the modern principles of design reuse.
- Concurrent engineering basic rules (unique EDA environment and common design database) are fulfilled.

The digital controller circuit design is currently being synthesised. XILINX FPGA implementation is targeted. The extensive use of VHDL for drive systems modelling and simulation, in conjunction with accurate digital controllers design is foreseen for the near future.

7. References

[1] I. M. Gottlieb, 'Electric Motors and Control Techniques', McGraw Hill, 1994.

[2] R. M. Crowder, 'Electric Drives and their Controls', Oxford University Press Inc, 1995.

[3] T. Katsunori, O. Yasumasa and I. Hisaichi, 'PWM Technique For Power MOSFET Inverter', IEEE Trans. on power electronics, Vol.3, No.3, 1998, pp.328-334.

[4] M. David and W. Donald, 'Current Control of VSI-PWM Inverters', IEEE Trans. on power electronics, vol. IA-21, No 4, May/June. 1985, pp. 562-570.

[5] A. Nabae, S. Ogasaward and H. Akagi, 'A Novel Control Scheme for Current Controlled PWM Inverters', IEEE Trans. on power electronics, vol. IA-22, No 4, July/August. 1986, pp. 697-701.

[6] D. Perry, - "VHDL", McGraw-Hill, 1998.

[7] Z. Navabi. 'VHDL: Analysis and Modeling of Digital Systems', McGraw Hill,1998.

[8] K. Tazi and E. Monmasson, 'Single-Chip DSP Based Speed Control Of Two AC-Machine', Speedam, Sorrento (Italy),3-5 june 1998, pp. P4-33 to P4-38.

[9] P. Vas, 'Vector Control of AC Machine', Clarendon Press, Oxford, UK, 1990.

Appendix A - VHDL Motor Model File

```
LIBRARY math;
USE math.mathtyx.all;
USE std.textio.all;
-- Electrical+mechanical model
ENTITY motor IS
  PORT (vds,vqs,Tl: IN Real;
        ids,iqs,wr: OUT REAL);
END motor;
ARCHITECTURE arch_motor OF motor IS
  CONSTANT Rs: REAL := 5.9;
  CONSTANT Rr: REAL := 4.62;
  CONSTANT ls: REAL := 0.831;
  CONSTANT lr: REAL := 0.833;
  CONSTANT lm: REAL := 0.809;
  CONSTANT jr: REAL := 0.001;
  CONSTANT deltat:TIME:= 1000ns;
  CONSTANT dt: REAL :=1.0e-6;
  CONSTANT wc: REAL := 50.0;
  CONSTANT p: REAL := 4.0;
  CONSTANT FG: REAL := 0.05;
  constant flag:integer:=1;
-- ***  Speed controller  ***
  CONSTANT ki: REAL :=0.05;
  CONSTANT kp: REAL :=1;
-- *** Torque Controller ***
  CONSTANT kit: REAL :=0.01;
  CONSTANT kpt: REAL :=1;
-- *** iy Current controller ***
  CONSTANT kci: REAL :=3;
  CONSTANT kcp: REAL :=100;
-- *** ix current controller ***
  CONSTANT kixi: REAL :=3.8;
  CONSTANT kixp: REAL :=100.5;
-- *** Flux controller ***
  CONSTANT kmci: REAL :=3.0;
  CONSTANT kmcp: REAL :=100.0;
CONSTANT imref: REAL :=0.005;
signal next_step: INTEGER := 1;
signal vdsr,vqsr: REAL :=0.0;
signal vdsr1,vqsr1: REAL :=0.0;
signal tt:real:=1.0;
FILE outf: TEXT IS OUT "C:\motor.txt";
BEGIN
PROCESS(next_step)
VARIABLE my_line: LINE;
VARIABLE a,ids1,idss,iqs1,iqss:
REAL:=0.0;
VARIABLE idr1,idrr,iqr1,iqrr:
REAL:=0.0;
VARIABLE Te,wr1,wrr: REAL :=0.0;
variable tetar,tr,imr,imr1,isx,fluxr,
         isy,wmr,tetamr :real:=0.0;
VARIABLE dif,difi,Trq,ert,teref,erti,
         isyref: REAL :=0.0;
```

```
VARIABLE cer,ceri,lss,vdx,vq,vsy,vdy:
        REAL :=0.0;
VARIABLE mce,mcei,isxref: REAL :=0.0;
VARIABLE cxe,cxei,vsx,vd: REAL :=0.0;
variable t:real:=0.0;
CONSTANT d_space: STRING :="   ";
BEGIN
    IF next_step=1 THEN
      WRITE(my_line,wc);
      WRITE (my_line,d_space);
      WRITE(my_line,wrr);
      WRITE (my_line,d_space);
      WRITE(my_line,vdsr);
      WRITE (my_line,d_space);
      WRITE(my_line,vds);
      WRITE (my_line,d_space);
      WRITE(my_line,tt);
      WRITE (my_line,d_space);
      WRITE(my_line,t);
      WRITE (my_line,d_space);
      WRITELINE(outf,my_line);
    END IF;
    Tr:=rr/lr;
    a:=(lm*lm-lr*ls);
    IF tt =1.0 then
    vdsr1 <= vds;
    vqsr1 <= vqs;
    tt <= 0.0;
    else
    vdsr1 <= vdsr;
    vqsr1 <= vqsr;
    END IF;
ids1:=(rs*lr*idss-wrr*lm*lm*iqss-rr*
   lm*idrr-wrr*lr*lm*iqrr-lr*vdsr1)/a;
iqs1:=(wrr*lm*lm*idss+rs*lr*iqss+wrr*
    lr*lm*idrr-rr*lm*iqrr-lr*vqsr1)/a;
idr1:=-(rs*lm*idss-wrr*lm*ls*iqss-rr*
   ls*idrr-wrr*lr*ls*iqrr-lm*vdsr1)/a;
iqr1:=-(wrr*lm*ls*idss+rs*lm*iqss+
wrr*lr*ls*idrr-rr*ls*iqrr-lm*vqsr1)/a;
idss:=idss+(ids1*dt);
iqss:=iqss+iqs1*dt;
idrr:=idrr+idr1*dt;
iqrr:=iqrr+iqr1*dt;
Te:=p*(3.0/2.0)*lm*(iqss*idrr-idss*
      iqrr);
wr1:=(Te-Tl)/jr;
wrr:=wrr+wr1*dt;
tetar:=tetar+wrr*dt;
t:=t+dt;
-- End of electrical+mechanical model
-- *** Vector Control ***
     imr1:=(isx-imr)/Tr;
     imr:=imr+imr1*dt;
     fluxr:=lm*imr;
     if imr >0.0 then
        wmr:= wrr+(isy/(Tr*imr));
     end if;
     tetamr:=tetamr+wmr*dt;
     isx:=idss*cos(tetamr)+
          iqss*sin(tetamr);
     isy:=-idss*sin(tetamr)+
           iqss*cos(tetamr);
-- *** Speed control loop ***
     dif :=(wc-wrr);
     difi:=difi+dif*dt;
     teref:=(ki*difi+kp*dif);
-- *** Actual Torque   ***
     Trq := p*(3.0/2.0)*
            (lm*lm/lr)*imr*isy;

-- *** Torque control ***
     ert:=teref-trq;
     erti:=erti+ert*dt;
     isyref:=(kit*erti+kpt*ert);
-- *** iy current control ***
     cer:= isyref-isy;
     ceri:=ceri+cer*dt;
     vsy:=(kci*ceri+kcp*cer);
-- *** Decoupling block ***
lss:=ls-(lm*lm/lr);
vdy:=wmr*lss*isx+(ls-lss)*wmr*imr;
vdx:=-wmr*lss*isy;
vq:=vsy+vdy;
-- *** Flux controller ***
     mce:=imref-imr;
     mcei:=mcei+mce*dt;
     isxref:=(kmci*mcei+kmcp*mce);
-- *** ix current controller  ***
     cxe:=isxref-isx;
     cxei:=cxei+cxe*dt;
     vsx:=(kixi*cxei+kixp*cxe);
     vd:=vsx+vdx;
-- ******************************
vdsr <= vd*cos(tetamr)-vq*sin(tetamr);
vqsr <= vd*sin(tetamr)+vq*cos(tetamr);
-- ******************************
IF next_step<500 THEN
next_step<=next_step+1 AFTER deltat;
ELSE
next_step<=1 AFTER deltat;
END IF;
ids<=idss;
iqs<=iqss;
wr<=wrr;
END PROCESS;
END arch_motor;
CONFIGURATION conf_motor OF motor IS
  FOR arch_motor
  END FOR;
END conf_motor;
```

DRIVES AND CONTROLS 2001
EXHIBITION & CONFERENCE
13-15 MARCH 2001
ExCeL London

POWER ELECTRONICS

HYDRAULICS & PNEUMATICS 2001 EXHIBITION

2001 Drives and Controls and Power Electronics

conference session proceedings

13 - 15 March 2001, Excel Exhibition Centre, London

Session 2

Power in Motion

Precision Motion

DRIVES AND CONTROLS 2001 EXHIBITION & CONFERENCE

Session 2

Chairman: Professor Barrie Jones, Aston University, UK

Tuesday, 13 March

9.00 - 9.30	REGISTRATION	
9.30 - 9.40	Chairman's introductory remarks	
9.40-10.10	**Paper 1: New solutions for drive strategies for stepper motors** Hideo Dhomeki, Oriental Motor Co., Japan	**1**
10.10-10.40	**Paper 2: A new Piezo electric motor with unlimited travel** Dr Giora Baum , Nanomotion, Germany	
10.40-11.00	COFFEE BREAK	
11.00-11.30	**Paper 3: Linear voice coil actuator technology is applied to linear brushless DC motors** Mikhail Godkin PhD, BEI Kimco, USA	**10**
11.30-12.00	**Paper 4: A new control IC for five phase step motor drive systems** Kouhei Mortaka, Oriental Motor Co. Japan, Kazuhiro Takahasi, Sanken Electric, Japan Dan Jones, Incremotion, USA,	**14**
12.00-2.15	LUNCH & VISIT EXHIBITION (This includes time to visit the Poster session)	
2.15-2.45	**Paper 5: Sensorless induction motor FPGA controller using polar co-ordinates** Andrei Dinu, , M N Cirstea, M McCormick, A Aounis, De Montfort University, UK	**19**
2.45-3.15	**Paper 6: The application of brushless motor drives to fast response servo systems** Dr Fred Garner, Professor M J Goodwin, Professor T Ruxton, Stafford University, UK	**25**
3.15-3.30	TEA BREAK	
3.30-4.00	**Paper 7: Blending techniques** Mirko Borich, Ilan Cohen, Kollmorgen, Israel	**35**
4.00-4.30	**Paper 8: Analysis and design of an integrated electric drive train for a flywheel powered ultra light rail bus** J Wang, R Perryman, University of East London, UK	**40**
4.30-5.00	**Paper 9: DSP power gives new life to 30 year old linear encoder technology** Dr Mark Hudman, Geoff Glasgow, Ewan Lee, Newall, UK	**44**
5.00-5.30	**Paper 10: Enhanced software techniques for selection of high performance drive systems** P A Littlehales, Aston University, UK, J Durrant, Rockwell Automation, UK, G Singh, Compro Ltd, India	**50**
5.30	CLOSING REMARKS	

Drives and Controls Conference 2001
13-15 March 2001
ExCel Exhibition Centre London

Sensorless Induction Motor FPGA Controller using Polar Coordinates

A. Dinu Lecturer, M N Cirstea Senior Lecture
M McCormick Head of Graduate Studies, A Aounis PhD Studen
De Montfort University, Leicester LE1 9BH United Kingdo

Session 2 Paper 5

ABSTRACT

This paper presents a simplified sensorless induction motor control strategy suitable for inexpensive low complexity FPGA hardware implementation. The new control strategy involves polar co-ordinates instead of the classical d-q axes thereby simplifying the transformations between stator and rotor flux co-ordinates. Furthermore, the rotor flux orientation is not calculated in the classical manner but it is approximated using an equivalent three-phase R-L-e circuit of the motor. The new sensorless speed control method has been tested by computer simulations and validated by practical experiments involving a XC4010XL FPGA and a 1kW motor.

INTRODUCTION

The large industrial use of induction motors has been stimulated over the years by their relatively low prices, low maintenance cost and high reliability. Consequently, a large number of induction motor control strategies have been developed [1]. The common approaches to the physical implementation of these control methods are the use of Digital Signal Processors (DSPs), Application Specific Integrated Circuits (ASICs), Field Programmable Logic Arrays (FPGAs) or a combination of them.

Attempts have already been made to involve FPGAs in the design of motor controllers but they were not very efficient because several circuits needed to be used. For instance, the strategy presented in [2] requires the implementation of a custom mathematical processor alongside with specialised control modules, and memory modules. The present paper analyses a new control strategy and demonstrates that it can be implemented as a single-FPGA controller thereby increasing the reliability of the system and simplifying both the design and the manufacturing process of the controller. The control strategy is devised in a manner that minimises the hardware implementation complexity and is appropriate for applications where extremely fast speed changes are unnecessary (fans, pumps, automotive).

THE SENSORLESS SPEED CONTROL STRATEGY

The principles underlying the new sensorless spee control strategy are derived from a simplified R-L model of the induction motor. Thus, the motor described by an equivalent three-phase circuit having resistor R, an inductor L and a voltage source e on eac phase. The behaviour of the R-L-e circuit is described b the space-vector equation

$$\underline{u} = R\underline{i} + L\frac{d\underline{i}}{dt} + \underline{e} \qquad (1$$

The equivalent circuit parameters, are related to th motor in the manner illustrated by equations (2), whic can be readily demonstrated using the space vect model of the induction motor expressed in stator co ordinates.

$$\begin{aligned} L &= \frac{L_s L_r - L_m^2}{L_r} \\ R &= R_s \\ \underline{e} &= \frac{L_m}{L_r}\left[-R_r \underline{i}_r^s + j\omega_{er}\left(L_r \underline{i}_r^s + L_m \underline{i}_s^s\right)\right] \\ \underline{u} &= \underline{u}_s \\ \underline{i} &= \underline{i}_s \end{aligned} \qquad (2$$

The common sensorless induction motor contr strategies are derived from the sensor-based fie oriented control methods, which have been extended t include speed estimation algorithms [4]. All fie orientation methods require several transformations o the electromagnetic quantities from the stator referenc frame into the flux reference frame, and back from th flux reference frame into stator reference frame. Th implies matrix multiplications with vectors (2x2 matrice and 2x1 vectors) and hence it necessitates a larg amount of hardware resources. The calculatio complexity involved by the control process can be muc reduced by using polar co-ordinates instead o rectangular co-ordinates because the space vecto modules and the phase shifts between space vectors ar invariant to reference frame transformations. Howeve

for the following mathematical investigations, both the rectangular and the polar reference frames need to be used.

The proposed control strategy is first analysed for steady state and very slow transient operation and then it is extended to fast transient operation. The steady-state control is summarised by three basic principles:

- The motor operation is controlled by means of stator current frequency fs and stator current amplitude I_s.
- The motor slip frequency $f_{slp}=f_{es}-f_{er}$ is maintained constant at a reference value $F_{slp}=\Omega_{slp}/2\pi$ by modifying the stator current amplitude accordingly.
- The slip information is extracted by analysing the phase shift between space vectors e and i in the equivalent R-L-e circuit.
- As a consequence of the previous principles, the motor speed can be controlled in steady state or slow transient operation by calculating the stator frequency according to the linear equation

$$f_s=\frac{p}{2\pi}\cdot\omega_{es}=\frac{p}{2\pi}\cdot\left(\omega_{er}^{ref}+\omega_{slp}\right)=\frac{p}{2\pi}\cdot\left(\omega_{es}^{ref}+\Omega_{slp}\right) \quad (3)$$

where p is the number of pairs of poles.

2.1 The slip estimation method

A rectangular reference frame oriented by the stator current is used to demonstrate the effect of the motor slip frequency over the phase shift between vectors $\underline{e}$ and $\underline{i}$ in steady state. The reference frame is rotating with the synchronism speed so that the stator current always lies along the real axis and therefore it can be expressed as a real quantity I_s. The general induction motor space vector model in the synchronous rotating reference frame is described by system (4) while the differential equation governing the rotor current vector is (5).

$$\begin{cases}\underline{u}_s^{syn}=R_sI_s+\dfrac{d\underline{\Psi}_s^{syn}}{dt}+j\omega_{es}\underline{\Psi}_s^{syn}\\ 0=R_r\underline{i}_r^{syn}+\dfrac{d\underline{\Psi}_r^{syn}}{dt}+j(\omega_{es}-\omega_{er})\underline{\Psi}_r^{syn}\\ \underline{\Psi}_s^{syn}=L_sI_s+L_m\underline{i}_r^{syn}\\ \underline{\Psi}_r^{syn}=L_r\underline{i}_r^{syn}+L_mI_s\end{cases} \quad (4)$$

$$\frac{d\underline{i}_r^{syn}}{dt}+\left(\frac{R_r}{L_r}+j\omega_{slp}\right)\underline{i}_r^{syn}=-\frac{L_m}{L_r}\left(j\omega_{slp}I_s+\frac{dI_s}{dt}\right) \quad (5)$$

In steady-state operation, the motor currents are sinusoidal and have constant frequency, which entails circular trajectories for the corresponding space vectors. Thus, the rotor current space vector is:

$$\underline{i}_r^{syn}=\frac{-j\omega_{slp}L_mI_s}{R_r+j\omega_{slp}L_r} \quad (6)$$

The internal voltage vector $\underline{e}^{syn}$ is determined by substituting (6) in (2). Calculating the real and the imaginary part of $\underline{e}^{syn}$ in steady-state yields

$$\begin{cases}\mathrm{Re}\left\{\underline{e}^{syn}\right\}=\dfrac{\omega_{es}\omega_{slp}L_mL_rR_rI_s}{R_r^2+\omega_{slp}^2L_r^2}\cdot\dfrac{L_m}{L_r}\\ \mathrm{Im}\left\{\underline{e}^{syn}\right\}=\dfrac{\omega_{es}L_mR_r^2I_s}{R_r^2+\omega_{slp}^2L_r^2}\cdot\dfrac{L_m}{L_r}\end{cases} \quad (7)$$

Therefore, the slip angular frequency (slp is linearly related to the real-to-imaginary ratio:

$$\tan^{-1}\left(\arg(\underline{e}^{syn})-\arg(\underline{i}^{syn})\right)=\frac{\mathrm{Re}\left\{\underline{e}^{syn}\right\}}{\mathrm{Im}\left\{\underline{e}^{syn}\right\}}=\frac{\omega_{slp}L_r}{R_r} \quad (8)$$

The difference between the two vector arguments is independent of the reference frame. Therefore, it is more practical to calculate the slip angular frequency in the stator reference frame:

$$\omega_{slp}=\tan^{-1}\left[\arg\left(\underline{e}^{s}\right)-\arg\left(\underline{i}_s^{s}\right)\right]\cdot\frac{R_r}{L_r} \quad (9)$$

2.2 The slip control method

For a given stator current amplitude I_s, the locus of the vector $\underline{e}^{syn}$ in the complex plane is a pair of circles defined by (10). The circles are tangent to the real axis in the origin of the reference frame (1).

$$\begin{cases}\left(\mathrm{Re}\{\underline{e}^{syn}\}\right)^2+\left(\mathrm{Im}\{\underline{e}^{syn}\}-R\right)^2=R^2\\ R=\dfrac{L_m^2}{2L_r}\cdot|\omega_{es}|\cdot I_s\end{cases} \quad (10)$$

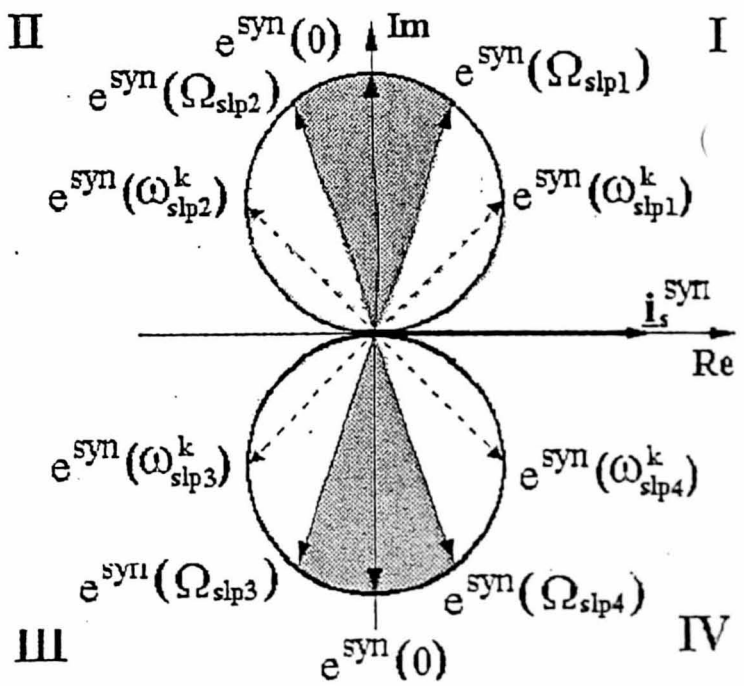


Fig.1. The locus of $\underline{e}^{syn}$ in the complex plane.

Each quadrant in Fig. 1 corresponds to a combination of the stator angular frequency sign and the slip angular frequency sign. The stator angular frequency is positive in quadrants I and II and negative in III and IV. For each

quadrant there is one reference slip angular frequency Ω_{slp}. The reference values Ω_{slp1}, Ω_{slp2}, Ω_{slp3}, and Ω_{slp4} have equal absolute values but different signs.

The calculations referring to four quadrants can be reduced to equivalent calculations in only one quadrant due to the symmetry of the diagram. The transformation from four quadrants to one is carried out by replacing the real and imaginary parts of vector e^{syn} with their absolute values. The result is an equivalent voltage calculated according to

$$\underline{E}_{eqv}^{syn} = \left|Re\left\{\underline{e}^{syn}\right\}\right| + j \cdot \left|Im\left\{\underline{e}^{syn}\right\}\right| \qquad (11)$$

Maintaining a constant slip angular frequency is equivalent to keeping the angle aeqv (Fig. 2). The relation between the reference slip angular frequency Ω_{slp} and the reference angle is defined in equation (12). Thus, the derivative of the stator current amplitude is calculated as a function of the error angle (β_{eqv}) illustrated in Fig. 2. The operation of the slip control loop is described in its simplest form by (13) where K_I is a positive constant.

$$\alpha_{eqv}^{ref} = \arctan\left(\frac{\Omega_{slp} L_r}{R_r}\right) \qquad (12)$$

$$\begin{cases} \dfrac{dI_s}{dt} = \begin{cases} K_I \cdot \beta_{eqv} & \text{when } I_s \in (I_{s-min}; I_{s-max}) \\ 0 & \text{when } I_s \notin (I_{s-min}; I_{s-max}) \end{cases} \\ \beta_{eqv} = \arg\left\{\underline{E}_{eqv}^{syn}(\Omega_{slp})\right\} - \arg\left\{\underline{E}_{eqv}^{syn}(\omega_{slp})\right\} \end{cases} \qquad (13)$$

Therefore, the stator current amplitude is a means of controlling the motor slip. Increasing the stator current increases the torque, which entails a decrease of the slip angular frequency. Conversely, the slip can be increased by decreasing the stator current amplitude. For practical reasons, the current amplitude is maintained between acceptable limits (I_{s-min} and I_{s-max}). The lower limit is necessary to insure that the slip frequency can be calculated while the upper limit is set to protect the motor and the PWM inverter.

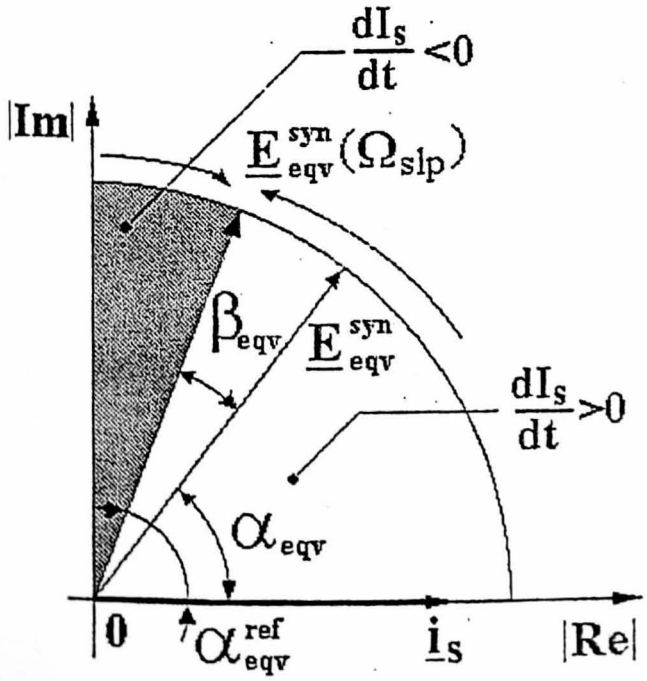


Fig. 2. The reduction of the four quadrants to one and the slip control method.

2.3. *Extended control method for fast transients*

The relation (3) between the stator frequency and motor speed is linear in steady state operation because, according to the presented control algorithm, the slip angular frequency ω_{slp} is maintained constant. However, ω_{slp} cannot be correctly estimated using relation (9) during fast transient operation because all the electromagnetic quantities including the internal voltage e^{syn} undergo oscillations. These oscillations make the slip control difficult and limit the system dynamic performance. The transient response can be improved by using a quasi field-orientated control strategy without actually performing all the mathematical calculations normally required. Thus, the rotor flux orientation can be approximated based on the position of the internal voltage vector e^{syn}. The relation between the rotor flux ψ_r and the internal voltage e^{syn} is described by equation (14) which is derived from (2).

$$\underline{e}^{syn} = \frac{L_m}{L_r}\left[R_r \underline{i}_r^{syn} + j\omega_{es} \underline{\Psi}_r^{syn}\right] \qquad (14)$$

If the speed is larger than a few rotations per minute then e^{syn} is approximately perpendicular on the rotor flux vector ψ_r because the rotor resistance can be neglected as compared to motor reactance.

The field orientation solution is to control the rotor speed by altering the stator current component i_{sq} while keeping i_{sd} constant. According to the new control method, the same thing is approximately achieved by simultaneously changing the stator current angular frequency ω_{es} and the stator current amplitude I_s. If γ is the angle between the rotor flux and the stator current (Fig. 3) then the two stator current components can be written as:

$$\begin{cases} i_{sd} = I_s \cdot \cos\gamma \\ i_{sq} = I_s \cdot \sin\gamma \end{cases} \qquad (15)$$

The derivatives of the stator current components during transient are given by

$$\begin{cases} \dfrac{di_{sd}}{dt} = \dfrac{dI_s}{dt}\cos\gamma - I_s \sin\gamma \cdot \dfrac{d\gamma}{dt} = 0 \\ \dfrac{di_{sq}}{dt} = \dfrac{dI_s}{dt}\sin\gamma + I_s \cos\gamma \cdot \dfrac{d\gamma}{dt} \end{cases} \qquad (16)$$

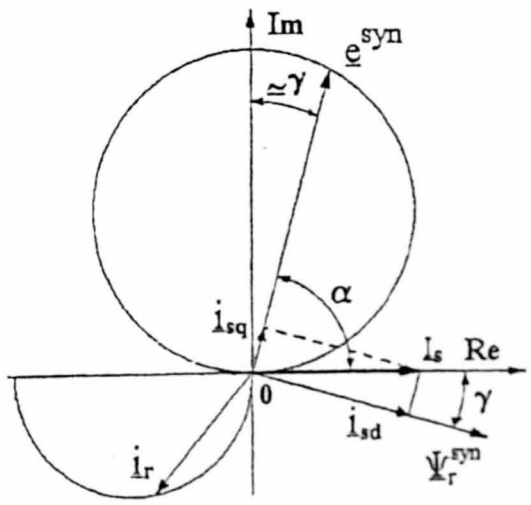


Fig. 3. The approximate position of vectors e, i and Ψ_r.

Equation (17) derived from the system (16) demonstrates the relation between the torque generating component i_{sq} and the angle γ for the rotor flux orientated vector control strategy.

$$\frac{di_{qs}}{dt}\cdot\frac{1}{I_s}\cdot\frac{1}{tg\gamma\cdot sin\gamma+cos\gamma}=\frac{d\gamma}{dt} \qquad (17)$$

It can be demonstrated that the angle γ undergoes damped oscillations during any of the induction motor. However, modifications of the stator frequency f_s create larger amplitude oscillations than the modifications of the stator current amplitude. On the other hand, if the $d\omega_{es}/dt$ is not excessively large and if constant K_1 in (13) is sufficiently small, then the transient is overdamped and $d\gamma/dt$ is approximately proportional with $d\omega_{es}/dt$. Therefore, good dynamic performance can be obtained without explicitly using field oriented vector control methods if ω_{es} is controlled in a manner that adequately takes into account the values of γ and I_s.

According to equation (17), $d\gamma/dt$ (and implicitly $d\omega_{es}/dt$), need to decrease with the increase of the angle γ. On the other hand any conclusion referring to $d\omega_{es}/dt$ is only valid if the stator frequency variations is not too fast. Consequently, the solution proposed here is mathematically expressed by (18) where $\alpha_{eqv}=\pi/2-\gamma_{eqv}$ and F is a non-linear function of α_{eqv} and I_s.

$$\begin{cases}\dfrac{d\omega_{es}}{dt}=sign\left(\omega_{er}^{ref}(t)+\Omega_{slp}-\omega_{es}\right)\cdot F\left(I_s,\alpha_{eqv}\right)\\ F\left(I_s,\alpha_{eqv}\right)>0\end{cases} \qquad (18)$$

This solution ensures that the variations of the stator current frequency are related with the values of parameters γ and I_s and in the same time $d\omega_{es}/dt$ is limited to values that validate the previous approximations. There are several acceptable versions of function F that approximate the vector control strategies with different degrees of accuracy. The simplest form of function F that generates reasonable control performance depends on the α_{eqv} alone

$$F\left(I_s,\alpha_{eqv}\right)=\begin{cases}K_1\cdot\alpha_{eqv}+K_2 & \text{if} \quad \alpha_{eqv}>\alpha_{min}\\ K_2 & \text{if} \quad \alpha_{eqv}\le\alpha_{min}\end{cases} \qquad (19)$$

where K_1, K_2 and α_{min} depend on the motor parameters and on the required dynamic performance of the drive system. The constant K_1 couples the current control loop with the slip control loop. When K_1 is larger than zero quasi field-oriented control strategy is obtained while $K_1=0$ leads to a quasi scalar control strategy. The adopted simple piecewise linear function (illustrated in Fig. 4) has the advantage that it minimises the hardware implementation complexity.

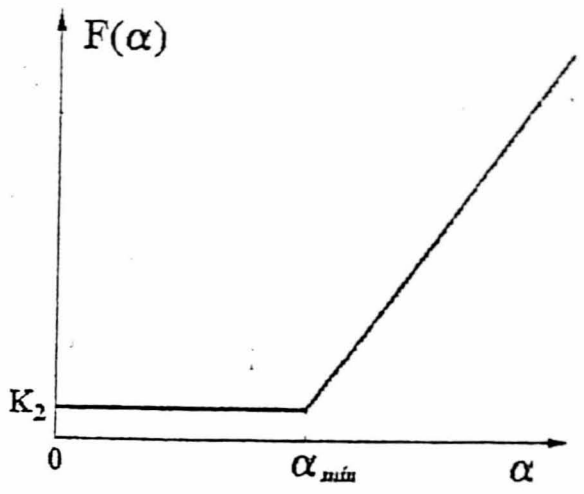


Fig. 4. Piecewise linear version of function F(I_s, α_{eqv})

The speed control strategy can be further refined by replacing the simple expression in (13) with a function depending both on beqv and on the stator current amplitude I_s. A constant K_1 cannot generate optimal results for all the motor currents in the range ($I_{s\text{-}min}$; $I_{s\text{-}max}$). The motor steady-state torque is proportional to I_s^2 so that the same derivative dI_s/dt produces different torque variations for different values of I_s as demonstrated by relations (20).

$$T=f\left(\beta_{eqv}\right)\cdot I_s^2\Rightarrow\frac{dT}{dt}=2I_s\cdot f\left(\beta_{eqv}\right)\cdot\frac{dI_s}{dt} \qquad (20)$$

The effect is a slow dynamic response of the motor when the current is close to $I_{s\text{-}min}$ and a very fast one when the current is close to $I_{s\text{-}max}$. Thus, to optimise the motor response, relation (13) can be replaced by (21). This requires that the motor derivative does not depend on the stator current amplitude.

$$\frac{dI_s}{dt}=\begin{cases}K_1\cdot\beta_{eqv}/I_s & \text{if} \quad I_s\in\left(I_{s-min};I_{s-max}\right)\\ 0 & \text{if} \quad I_s\notin\left(I_{s-min};I_{s-max}\right)\end{cases} \qquad (21)$$

This improved control strategy insures the same dynamic parameters both at small stator currents and at large stator currents because the torque derivative is not dependent on the stator current amplitude (see (22)).

$$\frac{dT}{dt}=2I_s\cdot f\left(\beta_{eqv}\right)\cdot\frac{dI_s}{dt}=2K_1\cdot\beta_{eqv} \qquad (22)$$

In conclusion, the complete sensorless induction motor control algorithm developed here comprises a speed control procedure described by equations (18) and (19) and a current control procedure expressed by (21). These equations are less complex than a typical sensorless vector control model and therefore they are easier to implement into hardware.

3. VHDL Modelling and Simulation

The complete drive system including the analogue and the digital elements was modelled, simulated, evaluated and implemented using VHDL [3]. Thus, the VHDL code developed includes the digital controller a simplified PWM inverter model and a motor model based on the standard space vector equations. The integration of the

differential motor equations was carried out using Euler's method. This modelling approach provides important advantages because the design is compatible with many CAE software tools, a large range of implementation technologies are available and the VHDL controller modules can be reused for other applications. To verify the controller model, VHDL simulations of the complete system were performed and the resulting numerical values were exported to MATLAB.

The operation of the complete sensorless induction motor drive system has been tested and validated by simulation for a large variety of parameters and conditions. The simulation results in Fig. 5 and Fig. 6 refer to a two-pole 11 kW induction motor with the following parameters: R_s=0.371 Ω, R_r=0.415 Ω, $L_{s\sigma}$=2.72 mH, $L_{r\sigma}$=3.3 mH, L_m=84.33 mH, J=0.02 kg◊m². The presented results compare the behaviour of the same induction motor when controlled by a quasi field-oriented control strategy (K_2>0) and by a quasi-scalar control strategy (K_2=0). The superiority of the former compared to the latter is demonstrated by the much shorter transients that it is capable to generate.

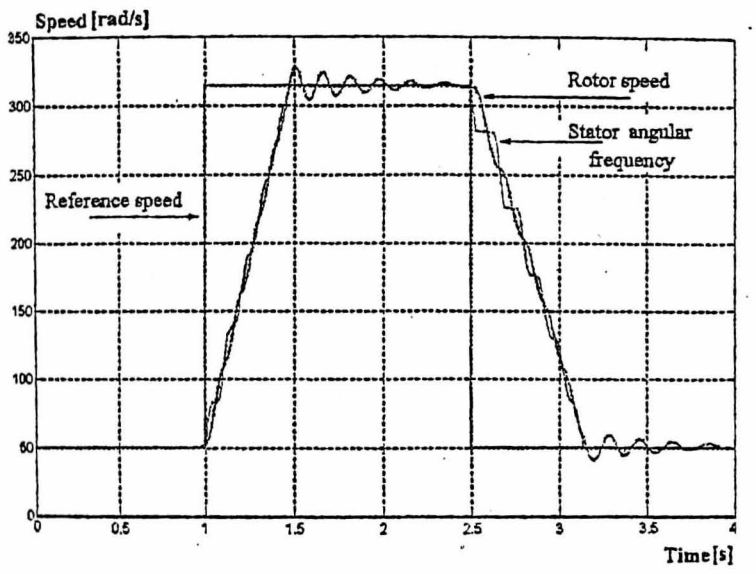


Fig. 5. Quasi field-oriented control (K_1=1000 s⁻¹, K_2=200 s⁻¹)

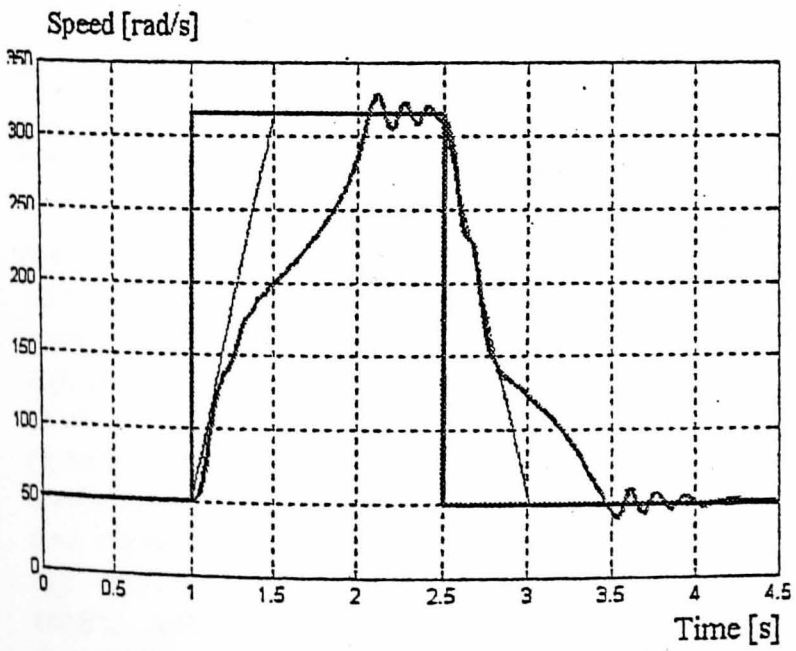


Fig. 6. Quasi-scalar control (K_1=0, K_2=200 s⁻¹)

In Fig. 5 the control system automatically finds an acceptable average acceleration by calculating $d\omega_{es}/dt$ as indicated in (18) correlated with (19). Faster transier may be obtained if F depends both on I_s and α_{es}, at th expense of increasing the hardware implementatic complexity. The very slow motor acceleration ar deceleration in Fig. 6 is due to the lack of correlatic between the two control loops conducing t inappropriate values for $d\omega_{es}/dt$ which 'force' the syste to generate a dynamic response beyond its limits. Th phenomenon can be eliminated by using a slightly smalle value for K_2 but the results cannot be improved to th level obtained using the quasi filed-oriented contr strategy.

4. Hardware Implementation and Practical Tests

FPGAs are ideal for shortening the design an development cycles and they allow the exploitation c the parallel processing enabled by hardwar implementation. The presented control strategy wa implemented in a Xilinx XC4010XL FPGA containin the equivalent of 10,000 logic gates. The hardwar structure obtained after the synthesis covers 99% of th chip and it operates at a clock frequency of 12 MHz.

The implementation is organised in four tier illustrated in Fig. 7. Tier0 generates a space vector o constant amplitude rotating with the variable angula speed indicated by the required to the stator curren vector. The space vector is given by its real and imaginar components that are generated using the differentia modulation technique to minimise the implementation. Thus, the look-up table included in Tier0 contains the differences between two consecutive samples instead of the samples themselves which allows generating 9-bit sinusoidal signals using reduced 3-bit entries. Tier1 multiplies the space vector generated by Tier0 with the reference amplitude of the stator current Is while Tier2 generates the corresponding switching pattern for the PWM inverter. The stator frequency control loop is implemented by Tier3. Low complexity hardware implemented neural networks are used for the calculation of the space vector arguments (relation (9)), for on-line estimation of the inductance L in the R-L-e circuit [5] and for the efficient current control [6], [7].

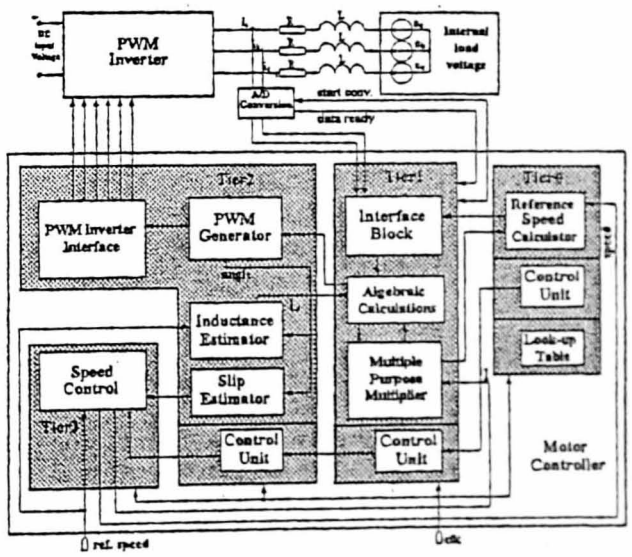


Fig. 7. The hardware implementation of the motor controller

Practical tests performed with a 1 kW motor validated the theoretical considerations underlying the new control approach. They are similar to the simulation results and prove the feasibility of the proposed FPGA controller. The ripples present in Fig. 8 and Fig. 9 are due to the imperfection of the measurement system used. They do not reflect real variations of the motor torque and speed.

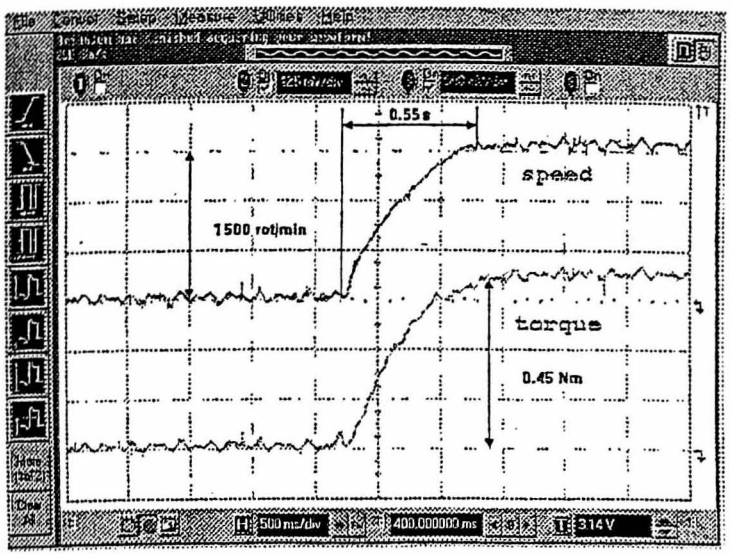


Fig. 8 –The drive system starting process

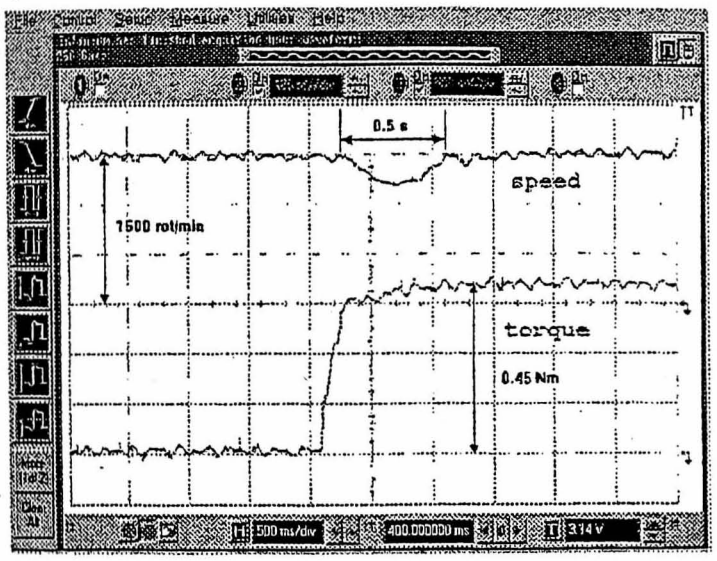


Fig. 9 –The step change of the load response

5. Conclusions

The main advantage of the quasi field-oriented control strategy presented in this paper is the low mathematical complexity, which simplifies the digital hardware implementation. The control quality obtainable with this approach is intermediate between the results of the sensorless scalar approach and those generated by sensorless field-oriented control. The proposed strategy is suitable to many industrial applications that require low cost implementations without the need for extremely high dynamic performance (fans, pumps). This speed control approach requires that vector e has an acceptable amplitude to allow the calculation of the angle α. As demonstrated by equation system (10) the amplitude of the internal load voltage is proportional to the stator angular frequency ω_{es} and to the stator current amplitude I_s. According to the presented theoretical considerations, the stator angular frequency wes is larger than zero at zero rotor speed: in this case it equals the reference slip angular frequency ($\omega_{es}=\Omega_{sp}$). Therefore, the calculations can be theoretically performed even at zero rotor speed. Practically, the control quality at low speeds depends on the number of bits generated by the A/D converters and the width of the internal busses of the controller.

The most important parameters affecting the control algorithm precision are the rotor resistance and the stator resistance. The rotor resistance variations directly affect the slip calculations (see equation (9)) based on vector e. On the other hand, the stator resistance variations affect the precision in calculating the vector e itself (the calculation is based on equation (1)). An increased rotor resistance leads to a negative speed error in steady-state operation while increased stator resistance generates a positive speed error. Both resistances increase simultaneously during the motor operation, and therefore the two errors partially cancel out. However, the implementation of a simple resistance estimator is envisaged by authors to increase the speed calculation accuracy. The estimator will use I_s and the rotor speed ω_r as inputs because they are the main quantities that determine the heat generation and the heat dispersion respectively.

References

[1] Leonhard, W.: "Control of electrical drives", 2nd edition Springer, Berlin 1996.

[2] Foussier, P., Calmon, F., Carrabina, J., Fathallah, M., Grennerat, V., Jorda, X., Gontrand, C., Retif, M.J., Chante, J.-P.: "Practical Example of Algorithm Integration for Electrical Drives. 8th European Conference on Power Electronics and Applications", EPE'99, 7-9 September 1999, Lausanne, Switzerland (CD).

[3] D. Perry, D.: "VHDL", McGraw-Hill, 1998.

[4] Vas, P.: "Sensorless Vector and Direct Torque Control", Monographs in Electrical and Electronic Engineering, Oxford University Press, 1998.

[5] Dinu A.; Cirstea M.N.; McCormick M; Ometto Antonio; Rotondalle Nicola: "Load Independent Current Control Strategy for PWM Inverters" in the proceedings of UKACC International Conference on Control (CONTROL'98) - September 1-4 1998 Swansea UK pp. 1118-1122.

[6] Dinu, A., Cirstea, M.N., McCormick, M: "Virtual Prototyping of a Digital Neural Current Controller". Ninth International Workshop on Rapid System Prototyping, June 3-5 1998, Leuven Belgium, p. 176-180.

[7] Dinu A.; Cirstea M.N.; McCormick M; Ometto A.; Rotondalle N.: "Neural Network for Control of PWM Inverters" in the proceedings of Power Electronics and Motion Control (PEMC'98), Prague, September 8 1998, CDROM.

Appendix A

```
**********************************************************************************
**********************************************************************************
**********                    VHDL Code                    **********
**********************************************************************************
**********************************************************************************
LIBRARY math;
USE math.mathtyx.all;
USE std.textio.all;
-- Electrical+mechanical model
ENTITY motor IS
  PORT (vds,vqs,Tl: IN Real;
        ids,iqs,wr: OUT REAL);
END motor;
ARCHITECTURE arch_motor OF motor IS
  CONSTANT Rs: REAL := 5.9;
  CONSTANT Rr: REAL := 4.62;
  CONSTANT ls: REAL := 0.831;
  CONSTANT lr: REAL := 0.833;
  CONSTANT lm: REAL := 0.809;
  CONSTANT jr: REAL := 0.001;
  CONSTANT deltat:TIME:= 1000ns;
  CONSTANT dt: REAL :=1.0e-6;
  CONSTANT wc: REAL := 50.0;
  CONSTANT p: REAL := 4.0;
  CONSTANT FG: REAL := 0.05;
  constant flag:integer:=1;
-- ***  Speed controller  ***
  CONSTANT ki: REAL :=0.2;
  CONSTANT kp: REAL :=1.2;
-- *** Torque Controller ***
  CONSTANT kit: REAL :=0.2;
  CONSTANT kpt: REAL :=1.2;
-- *** iy Current controller ***
  CONSTANT kci: REAL :=2.0;
  CONSTANT kcp: REAL :=12.5;
-- *** ix current controller ***
  CONSTANT kixi: REAL :=0.8;
  CONSTANT kixp: REAL :=160.5;
-- *** Flux controller ***
  CONSTANT kmci: REAL :=100.0;
  CONSTANT kmcp: REAL :=2000.0;
CONSTANT imref: REAL :=0.005;
signal next_step: INTEGER := 1;
signal vdsr,vqsr: REAL :=0.0;
signal vdsr1,vqsr1: REAL :=0.0;
signal tt:real:=1.0;
```

```
FILE outf: TEXT IS OUT "C:\motor.txt";
BEGIN
PROCESS(next_step)
VARIABLE my_line: LINE;
VARIABLE a,ids1,idss,iqs1,iqss: REAL:=0.0;
VARIABLE idr1,idrr,iqr1,iqrr: REAL:=0.0;
VARIABLE Te,wr1,wrr: REAL :=0.0;
variable tetar,tr,imr,imr1,isx,fluxr,
         isy,wmr,tetamr :real:=0.0;
VARIABLE dif,difi,Trq,ert,teref,erti,
         isyref: REAL :=0.0;
VARIABLE cer,ceri,lss,vdx,vq,vsy,vdy:
         REAL :=0.0;
VARIABLE mce,mcei,isxref: REAL :=0.0;
VARIABLE cxe,cxei,vsx,vd: REAL :=0.0;
variable t:real:=0.0;
CONSTANT d_space: STRING :="  ";
BEGIN
    IF next_step=1 THEN
      WRITE(my_line,wc);
      WRITE (my_line,d_space);
      WRITE(my_line,wrr);
      WRITE (my_line,d_space);
      WRITE(my_line,vdsr);
      WRITE (my_line,d_space);
      WRITE(my_line,vds);
      WRITE (my_line,d_space);
      WRITE(my_line,tt);
      WRITE (my_line,d_space);
      WRITE(my_line,t);
      WRITE (my_line,d_space);
      WRITELINE(outf,my_line);
    END IF;
    Tr:=Lr/rr;
    a:=(lm*lm-lr*ls);
    IF tt =1.0 then
    vdsr1 <= vds;
    vqsr1 <= vqs;
    tt <= 0.0;
    else
    vdsr1 <= vdsr;
    vqsr1 <= vqsr;
    END IF;
ids1:=(rs*lr*idss-wrr*lm*lm*iqss-rr*
   lm*idrr-wrr*lr*lm*iqrr-lr*vdsr1)/a;
iqs1:=(wrr*lm*lm*idss+rs*lr*iqss+wrr*
    lr*lm*idrr-rr*lm*iqrr-lr*vqsr1)/a;
idr1:=-(rs*lm*idss-wrr*lm*ls*iqss-rr*
   ls*idrr-wrr*lr*ls*iqrr-lm*vdsr1)/a;
```

```
iqr1:=-(wrr*lm*ls*idss+rs*lm*iqss+
wrr*lr*ls*idrr-rr*ls*iqrr-lm*vqsr1)/a;
idss:=idss+(ids1*dt);
iqss:=iqss+iqs1*dt;
idrr:=idrr+idr1*dt;
iqrr:=iqrr+iqr1*dt;
Te:=p*(3.0/2.0)*lm*(iqss*idrr-idss*
      iqrr);
wr1:=(Te-Tl)/jr;
wrr:=wrr+wr1*dt;
tetar:=tetar+wrr*dt;
t:=t+dt;
-- End of electrical+mechanical model
-- *** Vector Control ***
     imr1:=(isx-imr)/Tr;
     imr:=imr+imr1*dt;
     fluxr:=lm*imr;
     if imr >0.0 then
         wmr:= wrr+(isy/(Tr*imr));
     end if;
     tetamr:=tetamr+wmr*dt;
     isx:=idss*cos(tetamr)+
          iqss*sin(tetamr);
     isy:=-idss*sin(tetamr)+
          iqss*cos(tetamr);
-- *** Speed control loop ***
     dif :=(wc-wrr);
     difi:=difi+dif*dt;
     teref:=(ki*difi+kp*dif);
-- *** Actual Torque   ***
     Trq := p*(3.0/2.0)*
            (lm*lm/lr)*imr*isy;

-- *** Torque control ***
     ert:=teref-trq;
     erti:=erti+ert*dt;
     isyref:=(kit*erti+kpt*ert);
-- *** iy current control ***
     cer:= isyref-isy;
     ceri:=ceri+cer*dt;
     vsy:=(kci*ceri+kcp*cer);
-- *** Decoupling block ***
lss:=ls-(lm*lm/lr);
vdy:=wmr*lss*isx+(ls-lss)*wmr*imr;
vdx:=-wmr*lss*isy;
vq:=vsy+vdy;
```

```
-- *** Flux controller ***
      mce:=imref-imr;
      mcei:=mcei+mce*dt;
      isxref:=(kmci*mcei+kmcp*mce);
-- *** ix current controller  ***
      cxe:=isxref-isx;
      cxei:=cxei+cxe*dt;
      vsx:=(kixi*cxei+kixp*cxe);
      vd:=vsx+vdx;

-- ******************************
vdsr <= vd*cos(tetamr)-vq*sin(tetamr);
vqsr <= vd*sin(tetamr)+vq*cos(tetamr);
-- ******************************

IF next_step<500 THEN
next_step<=next_step+1 AFTER deltat;
ELSE
next_step<=1 AFTER deltat;
END IF;
ids<=idss;
iqs<=iqss;
wr<=wrr;
END PROCESS;
END arch_motor;

CONFIGURATION conf_motor OF motor IS
  FOR arch_motor
  END FOR;
```

Appendix B

```
**********************************************************************************
**********************************************************************************
**              The structural description of the FOC                           ***
**********************************************************************************
**********************************************************************************
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_signed.ALL;
-- ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
-- Main Program
-- ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
ENTITY main51 IS
 PORT(
 --      ia,ib:in std_logic_vector (8 downto 0);
--       qa,qb:in std_logic_vector (7 downto 0);
      imr,iqs,ids: out std_logic_vector(10 downto 0);
      imref:in std_logic_vector(10 downto 0);
      speed: in std_logic_vector(10 downto 0);
      refspeed: in std_logic_vector(10 downto 0);
      vsdreff:out std_logic_vector (10 downto 0);
      vsqreff:out std_logic_vector (10 downto 0);
      startt:out std_logic;
      Breset,Bclk:in std_logic;
      enable:in std_logic;
      start:inout std_logic;
      T1,T2,T3,T4,T5,T6:out std_logic;
      div:out std_logic_vector (10 downto 0));
END main51;
ARCHITECTURE main51_arch of main51 IS
--#####################################################
component testbench5 IS
 PORT(clk:in std_logic;
        signal startt:out std_logic;
        signal ia,ib:out  STD_LOGIC_VECTOR (7 downto 0)
 );
END component;
-- #####################################################
component clrktransform is
   port (
    ia,ib: in STD_LOGIC_VECTOR (7 downto 0);
    ialpha: out STD_LOGIC_VECTOR(8 downto 0);
    ibeta: out STD_LOGIC_VECTOR(8 downto 0)
     );
end component;
component smultiplier is
```

```
generic(
          n: in integer:=9;  -- The operand length
          m: in integer :=8;  -- The result length
          step_length:in integer:=1
         );
 port(  a:in std_logic_vector (8 downto 0);
          b:in std_logic_vector (7 downto 0);
          prod:out std_logic_vector (16 downto 0);
          clk,start:in std_logic;
          ready:out std_logic);
end component;
component ctr1 IS
 PORT( clk:in std_logic;
          readyc,readyc1,readyc2,readyc3 : in std_logic;
       ld1:out std_logic;
       ld2:out std_logic;
       ld3:out std_logic;
       ld4:out std_logic;
      startd:out std_logic);


END component;
component reg1 is
 PORT( ld11,reset : in std_logic;
       in_reg1:in std_logic_vector(9 downto 0);
       out_reg1:out std_logic_vector(9 downto 0));

end component;

component reg2 is
 PORT( ld12,reset : in std_logic;
       in_reg2:in std_logic_vector(9 downto 0);
       out_reg2:out std_logic_vector(9 downto 0));

end component;
component reg3 is
 PORT( ld13,reset : in std_logic;
       in_reg3:in std_logic_vector(9 downto 0);
       out_reg3:out std_logic_vector(9 downto 0));
end component;
component reg4 is
 PORT( ld14,reset : in std_logic;
       in_reg4:in std_logic_vector(9 downto 0);
       out_reg4:out std_logic_vector(9 downto 0));
end component;
component addr IS
 PORT( reset:in std_logic;
         out_reg11,out_reg12,out_reg13,out_reg14: in std_logic_vector (9 downto 0);
       ids:out std_logic_vector(10 downto 0);
```

```
      iqs:out std_logic_vector(10 downto 0));
END component;
component tst31 is
port ( clk,reset: in std_logic;
         ids:in std_logic_vector (10 downto 0);
         imr:out std_logic_vector (10 downto 0));
end component;
-- ##########################################
component divider is
  port (
     a: in STD_LOGIC_VECTOR (10 downto 0);
     b: in STD_LOGIC_VECTOR (10 downto 0);
     div: out STD_LOGIC_VECTOR (10 downto 0);--12
     clk,start: in STD_LOGIC;
     ready: out STD_LOGIC
   );
end component;
-- ##########################################

--component Dividera is
--   port (
 --       CLK: in std_logic;
 --       RESET: in std_logic;-
 --      dividend: in std_logic_vector(11 downto 0);   --(n-2 downto 0);--13
 --      divisor: in std_logic_vector(11 downto 0);     --(m-1 downto 0);--7
                  --       re:out std_logic;
--       result: out std_logic_vector(11 downto 0));   --(m-1 downto 0));--7
--end component;

-- *************** component Kt.const ****************
component dat2 is
    port (clk,reset:in std_logic;
      in2: in STD_LOGIC_VECTOR (10 downto 0);
      out2: out STD_LOGIC_VECTOR(10 downto 0));
end component;
-- *************** End component  ********************
component tetaa is
port ( clk,reset: in std_logic;
         speed:in std_logic_vector (10 downto 0);
         slip:in std_logic_vector (10 downto 0);
         teta:out std_logic_vector (10 downto 0));
end component;

-- ************************************************

component ctrm1 IS
 PORT( ready1 : in std_logic;
       startp:out std_logic
       );
```

```
END component;
-- ***************************************************
component sin_rom IS
 PORT( clk,reset,enable : in std_logic;
   A: IN std_logic_vector(10 DOWNTO 0);-- it was 10
   q:out std_logic_vector(7 downto 0);
   q1: out std_logic_vector(7 downto 0));
END component;
--***************************************************
component speed_regulator is
port ( readys,reset: in std_logic;
        speed:in std_logic_vector (10 downto 0); --10
        isq:in std_logic_vector (10 downto 0); --10
        isd:in std_logic_vector (10 downto 0); --10
        im:in std_logic_vector (10 downto 0); --10
        imref:in std_logic_vector (10 downto 0); --10
        refspeed:in std_logic_vector (10 downto 0);  --10
        isdref:out std_logic_vector (10 downto 0); --10
        vsdref:out std_logic_vector (10 downto 0); --10
        isqref:out std_logic_vector (10 downto 0);    -- it was 21
        vsqref:out std_logic_vector (10 downto 0));  -- it was 21
end component;
-- ###################################################
component main511 IS
 PORT(
        vsqref1,vsdref1:in std_logic_vector(10 downto 0);
      qa,qb: in std_logic_vector(7 downto 0);
      Breset,Bclk:in std_logic;
      startp:in std_logic;
      Tr1,Tr2,Tr3,Tr4,Tr5,Tr6:out std_logic);
END component;


-- ***************************************
signal ia,ib: std_logic_vector (7 downto 0);
signal qa,qb: std_logic_vector(7 downto 0);
signal ialph1,ibeta1: std_logic_vector(8 downto 0);
signal prodz,prodz1,prodz2,prodz3: std_logic_vector(16 downto 0);
signal outr11,outr12,outr13,outr14: std_logic_vector(9 downto 0);
signal vsqref1,vsdref1: std_logic_vector(10 downto 0);
signal im,ids1,iqs1: std_logic_vector(10 downto 0);
signal div1: std_logic_vector(10 downto 0);
signal slipf: std_logic_vector(10 downto 0);
signal tetam: std_logic_vector(10 downto 0);
signal readym,readym1,readym2,readym3,readyc,readyd,startp:std_logic;
signal ld11,ld12,ld13,ld14,ldd7,ldd8,ldd9,startdiv:std_logic;
signal  z,z1,z2,z3:std_logic_vector(9 downto 0);
signal  isdreff: std_logic_vector (10 downto 0);
```

```
signal  isqreff: std_logic_vector (10 downto 0);

begin

-- ***************
Tb:testbench5 port map(bclk,start,ia,ib);

-- ****************
PT:clrktransform port map(ia,ib,ialph1,ibeta1);          -- Clark transformation
-- **************      park transformation  ******************
u1:Smultiplier port map(ibeta1,qb,prodz,bclk,start,readym);   -- ibeta * cos.teta
u2:Smultiplier port map(ialph1,qa,prodz1,bclk,start,readym1); -- ialph * sin.teta
u3:Smultiplier port map(ialph1,qb,prodz2,bclk,start,readym2); -- ialph * cos.teta
u4:Smultiplier port map(ibeta1,qa,prodz3,bclk,start,readym3); -- ibeta * sin.teta
z<=prodz(16 downto 7);
z1<=prodz1(16 downto 7);
z2<=prodz2(16 downto 7);
z3<=prodz3(16 downto 7);
ct:ctr1 port map(bclk,readym,readym1,readym2,readym3,ld11,ld12,ld13,ld14,startdiv);--
readym blocked
re:reg1 port map(ld11,breset,z1,outr11);    --ld11
re2:reg2 port map(ld12,breset,z2,outr12);--ld12
re3:reg3 port map(ld13,breset,z3,outr13);--ld13
re4:reg4 port map(ld14,breset,z,outr14);--ld14
add:addr port map(breset,outr11,outr12,outr13,outr14,ids1,iqs1); -- to calculate ids and iqs

--*****************************************************
u6:tst31 port map(readym,breset,ids1,im);  -- readym instead of clk
                        --d1:dividera port map(clk,reset,iqs1,im,div1);
d1:divider port map(iqs1,im,div1,bclk,startdiv,readyd);
--
*************************************************************************
***********
u12:dat2 port map(readym,breset,div1,slipf);
u13:tetaa port map(readym,breset,speed,slipf,tetam);-- ldd7 instead of clk
u14:sin_rom port map(bclk,breset,enable,tetam,qa,qb);
u15:speed_regulator port
map(readym,breset,speed,iqs1,ids1,im,imref,refspeed,isdreff,vsdref1,isqreff,vsqref1);
-- ***************************************************
ctm:ctrm1 port map(readym,startp);--readym blocked
mm5: main511 port map(vsqref1,vsdref1,qa,qb,Breset,bclk,startp,T1,T2,T3,T4,T5,T6);
*************************************************************************
*************
ids<=ids1;
iqs<=iqs1;
imr<=im;
div<=div1;
startt<=startp;
vsqreff<=vsqref1;
```

```
vsdreff<=vsdref1;
end main51_arch;
```

::

:Clark-transform .vhd

::

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_signed.all;
use work.imp_pack.all;


entity clrktransform is

    port (
     ia,ib: in STD_LOGIC_VECTOR (7 downto 0);
     ialpha: out STD_LOGIC_VECTOR (8 downto 0);  -- The real part of the load current
vector
     ibeta: out STD_LOGIC_VECTOR (8 downto 0)    -- The imaginary part of load
current vector
      );
end clrktransform;

architecture clrktransform_arch of clrktransform is
                                    -- The real part of the load current vector
 signal ibeta1: STD_LOGIC_VECTOR (8 downto 0);     -- The imaginary part of load
current vector
  signal ibeta_par: STD_LOGIC_VECTOR (8 downto 0);  -- The partial result for
imaginary part before
                                    -- multiplication with 0.866
begin

-- ialpha<= ia;                                    --  Theoretically: ire=3/2*ia
-- ibeta1(0) <= ib(9);                                    --  Practically after rescaling:
ire=3/4*ia (on 11 bits)
-- ibeta1(9 downto 1)<=ib(8 downto 0);                --  Theoretically: iim_par=2*ib+ia
-- ibeta_par<= ibeta1+ia;

ialpha<=(ia(7) & ia);
ibeta_par<=(ib & '0')+ia;

 process(ibeta_par)    -- Multiplication by 0.57735=0.1001001111001\B
  variable rez: STD_LOGIC_VECTOR(20 downto 0);       -- 22 = 12+11-1
```

```
begin

 rez := ibeta_par & zeroes(12);
-- rez := rez - (ibeta_par & zeroes(11));
 rez := rez - (ibeta_par & zeroes(10));
 rez := rez - (ibeta_par & zeroes(9));
-- rez := rez - (ibeta_par & zeroes(8));
 rez := rez - (ibeta_par & zeroes(7));
 rez := rez - (ibeta_par & zeroes(6));
-- rez := rez - (ibeta_par & zeroes(5));
-- rez := rez - (ibeta_par & zeroes(4));
-- rez := rez - (ibeta_par & zeroes(3));
-- rez := rez - (ibeta_par & zeroes(2));
 rez := rez - (ibeta_par & zeroes(1));
 rez := rez - ibeta_par;
 ibeta<=rez(20 downto 12);
end process;
end clrktransform_arch;
```

::

Multiplier.vhd

::

```
        library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use work.imp_pack.all;

entity smultiplier is
   generic(n,m,step_length: integer);
  -- port (
  --  generic(
   --        n: in integer:=10;  -- The operand length
   --        m: in integer :=8;  -- The result length
   --        step_length:in integer:=1
   --      );
    port (
      a: in STD_LOGIC_VECTOR (n-1 downto 0); -- Can be only positive 9--0
      b: in STD_LOGIC_VECTOR (m-1 downto 0); -- Can be both positive and negative 9
      prod: out STD_LOGIC_VECTOR (n+m-1 downto 0);--17
   --  magid: out STD_LOGIC_VECTOR (n+m-1 downto 7);--17
      clk,start: in STD_LOGIC;
      ready: out STD_LOGIC
   );
   constant nsteps: integer := (m+step_length-1)/step_length;
   constant p: integer :=n+step_length*nsteps;
   constant ref: integer := step_length*(nsteps-1);
end smultiplier;

architecture smultiplier_arch of smultiplier is
```

```
 type state is (s0,s1,s2,s3,s4);
 signal s: state;
 signal int_prod: STD_LOGIC_VECTOR(p-1 downto 0);
-- signal hana: STD_LOGIC_VECTOR (n+m-1 downto 0);--17
 signal int_b: STD_LOGIC_VECTOR(m-1 downto 0);
 signal first_step,load_step,shift_step: STD_LOGIC;
 signal count: integer range 0 to nsteps;
 signal ready_int: STD_LOGIC;
begin

 process(a,b,int_prod)
 begin
  if b(m-1)='0' then
   prod<=int_prod(n+m-1 downto 0);
  --   prod<=hana;
  --    magid<=hana(n+m-1 downto 7);
   else
    prod<=int_prod(n+m-1 downto 0) - (a & zeroes(m));
  --   magid<=int_prod(n+m-1 downto 7);-- - (a );
  end if;
 end process;


 process(start,clk)
 begin
  if start='1' then
   int_prod<=(others=>'0');
  elsif clk='0' and clk'event then
   if first_step='1' then
    int_b<=b;
   elsif load_step='1' and ready_int='0' then
    int_prod(p-1 downto ref)<=int_prod(p-1 downto ref)+a*int_b(step_length-1 downto
0);
   elsif shift_step='1' and ready_int='0' then
    int_prod<=zeroes(step_length) & int_prod(p-1 downto step_length);
    int_b<=zeroes(step_length) & int_b(m-1 downto step_length);
   end if;
  end if;
 end process;

 process(clk,start)
 begin
  if start='1' then
   s<=s0;
   count<=nsteps;
  elsif clk='1' and clk'event then
   case s is
    when s0 => s<=s1;
    when s1 => s<=s2;
```

```
      when s2 => if count>1 then
                   s<=s3;
                   count<=count-1;
                 else
                   s<=s4;
                 end if;
      when s3 => s<=s2;
      when s4 => null;
      when others => null;
     end case;
   end if;
 end process;

 first_step <= '1' when s=s1 else '0';
 load_step  <= '1' when s=s2 else '0';
 shift_step <= '1' when s=s3 else '0';
 ready_int <= '1' when s=s4 else '0';
 ready<=ready_int;
end smultiplier_arch;
```

::

Ctr1.vhd

::

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_unsigned.ALL;

ENTITY ctr1 IS
 PORT( clk,readyc,readyc1,readyc2,readyc3 : in std_logic;

       ld1:out std_logic;
       ld2:out std_logic;
       ld3:out std_logic;
       ld4:out std_logic;
       startd:out std_logic);
      -- out_reg1:out std_logic_vector(19 downto 0)
END ctr1;
ARCHITECTURE ctr1_arch OF ctr1 IS
begin

 process (clk,readyc,readyc1,readyc2,readyc3)
  begin
  --**********************
-- if readyc='1' then
        if (clk='1'and clk'event ) then
         if readyc='1' then
          ld1<='1';
          startd<='0';
          else
```

```
            ld1<='0';
            startd<='1';
           end if;
           if readyc1='1' then
            ld2<='1';
            else
            ld2<='0';
           end if;
         if readyc2='1' then
            ld3<='1';
            else
            ld3<='0';
           end if;
          if readyc3='1' then
            ld4<='1';
            else
            ld4<='0';
           end if;
   end if;
  end process;
END ctr1_arch;
```

::

Reg1.vhd, Reg2.vhd, reg3.vhd and Reg4.vhd

::

```
 LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_unsigned.ALL;
ENTITY reg1 IS
 PORT( ld11,reset : in std_logic;
       in_reg1:in std_logic_vector(9 downto 0);
       out_reg1:out std_logic_vector(9 downto 0));

END reg1;
ARCHITECTURE reg1_arch OF reg1 IS
begin
 process (ld11,reset)
  begin

   if reset ='1' then
      out_reg1<=(others=>'0');          -- to intialize register q and q1 to 0
  elsif(ld11='1'and ld11'event ) then
        out_reg1 <= in_reg1;
       end if;

  end process;

END reg1_arch;
```

.....

```
 LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_unsigned.ALL;

ENTITY reg2 IS
  PORT( ld12,reset : in std_logic;
        in_reg2:in std_logic_vector(9 downto 0);
        out_reg2:out std_logic_vector(9 downto 0));
END reg2;

ARCHITECTURE reg2_arch OF reg2 IS

begin

  process (ld12,reset)
    begin

    if reset ='1' then
       out_reg2<=(others=>'0');          -- to intialize register q and q1 to 0
       elsif(ld12='1'and ld12'event ) then
       out_reg2 <= in_reg2;
   end if;

   end process;

END reg2_arch;
```

.....

```
 LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_unsigned.ALL;
ENTITY reg3 IS
  PORT( ld13,reset: in std_logic;
        in_reg3:in std_logic_vector(9 downto 0);
        out_reg3:out std_logic_vector(9 downto 0));
END reg3;
ARCHITECTURE reg3_arch OF reg3 IS
begin
  process (ld13,reset)
    begin
      if reset ='1' then
        out_reg3<=(others=>'0');          -- to intialize register q and q1 to 0
        elsif(ld13='1'and ld13'event ) then
        out_reg3 <= in_reg3;
   end if;
   end process;

END reg3_arch;
```

```
 LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_unsigned.ALL;
ENTITY reg4 IS
 PORT( ld14,reset: in std_logic;
      in_reg4:in std_logic_vector(9 downto 0);
      out_reg4:out std_logic_vector(9 downto 0));
END reg4;
ARCHITECTURE reg4_arch OF reg4 IS
begin
 process (ld14,reset)
  begin
   if reset ='1' then
    out_reg4<=(others=>'0');          -- to intialize register q and q1 to 0
    elsif(ld14='1'and ld14'event ) then
    out_reg4 <= in_reg4;
  end if;
  end process;
END reg4_arch;
```

Addr.vhd

```
 LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_signed.ALL;
use ieee.std_logic_arith.all;  -- i added this lately
ENTITY addr IS
 PORT( reset:in std_logic;
       out_reg11,out_reg12,out_reg13,out_reg14: in std_logic_vector (9 downto 0);
      ids:out std_logic_vector(10 downto 0);
      iqs:out std_logic_vector(10 downto 0));
END addr;
ARCHITECTURE addr_arch OF addr IS
begin
 process (reset,out_reg11,out_reg12,out_reg13,out_reg14)
  begin
  if reset ='1' then
    ids<=(others=>'0');
    iqs<=(others=>'0');
       else
    ids <= (out_reg12(9) & out_reg12)+out_reg13;
    iqs <= (out_reg14 (9) & out_reg14) -out_reg11;
  end if;
  end process;
END addr_arch;
```

Magnetizing-current (imr).vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;
use ieee.std_logic_arith.all;
use work.imp_pack.all;
entity tst31 is
port ( clk,reset: in std_logic;
        ids:in std_logic_vector (10 downto 0);
        imr:out std_logic_vector (10 downto 0));
end tst31;
architecture behavioral of tst31 is
signal int_imr:std_logic_vector(10 downto 0);
begin

 process (clk,reset)
          variable x:std_logic_vector (10 downto 0);
          variable cc:std_logic_vector (10 downto 0);
          variable rez: STD_LOGIC_VECTOR(23 downto 0);   -- 30 instead of 22
  begin
      if reset='1' then
      imr<= (others =>'0');
      elsif (clk='1' and clk'event) then
         x:=(ids-int_imr);
     -- **   0.00417 = 0.0000000100010    kr = T/Tr 100 e-6/0.024 = tr=0.024
        rez := rez + (x & zeroes(1))+(x & zeroes(5));
      cc  := rez(23 downto 13);
     int_imr<=int_imr+cc;
     imr<=cc;              --(9 downto 0);
    end if;
  end process;
end behavioral;
```

Divider.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_signed.all;
```

```
use work.imp_pack.all;
##################################################################################
-- This program uses only 11 bits for a and b, one of which is the sign
-- bit and the rest which are 10 bits are used for operand a and b
-- the total number is then 2^10 = 1204 = 400 (hexa) when dividing this number by 2
-- 1024/2= 512 (dec) --> 200 (hexa).
-- therefor 0 --> 199 used for the positive values.
-- and 0 --> 399 is used to represent the negative values.
--
##################################################################################
#######
entity divider is
   generic(
         n: in integer:=11;  -- The operand length
         m: in integer:=11   -- The result length
        );
   port (
      a: in STD_LOGIC_VECTOR (n-1 downto 0);
      b: in STD_LOGIC_VECTOR (n-1 downto 0);
      div: out STD_LOGIC_VECTOR (m-1 downto 0);
      clk,start: in STD_LOGIC;
      ready: out STD_LOGIC
   );
end divider;

architecture divider_arch of divider is
 signal par_rez: STD_LOGIC_VECTOR(n+m-1 downto 0);
 signal pos_a: STD_LOGIC_VECTOR(n-1 downto 0);
 signal pos_b: STD_LOGIC_VECTOR(n-1 downto 0);
 signal sub_rez: STD_LOGIC_VECTOR(n downto 0);
 signal res_sign,finish,prepare: STD_LOGIC;
begin


 process(start,clk)
  variable count: integer range 0 to 31;
 begin

  if start='1' then
   count:=m+2;
   finish<='0';
   prepare<='1';
  elsif clk='1' and clk'event then
   if finish='0' then
    prepare<='0';
    count:=count-1;
    if count=0 then
     finish<='1';
    end if;
```

```
   end if;
  end if;
end process;

process(a)
begin
  if a(n-1)='1' then
    pos_a<=-a;
  else
    pos_a<=a;
  end if;
end process;

process(b)
begin

  if b(n-1)='1' then
    pos_b<=-b;
  else
    pos_b<=b;
  end if;
end process;

sub_rez<=('0' & par_rez(m+n-1 downto m))-pos_b;

process(clk)
begin
  if clk='0' and clk'event then
    if prepare='1' then
      par_rez(m-1 downto m-n)<=pos_a;
      par_rez(m+n-1 downto m)<=zeroes(n);
      par_rez(m-n downto 0)<=zeroes(1);--        par_rez(m-n-1 downto 0)<=zeroes(m-n);
    elsif finish='0' and prepare='0' then
      if sub_rez(n)='0' then
        par_rez<=sub_rez(n-2 downto 0) & par_rez(m-1 downto 0) & '1';
      else
        par_rez<=par_rez(n+m-2 downto 0) & '0';
      end if;
    end if;
  end if;
end process;
res_sign<=a(n-1) xor b(n-1);
ready<=finish;
process(res_sign,par_rez)
begin
  if b="00000000000" then --
    div<="00000000000";--
   elsif res_sign='0' then--  if
    div<=par_rez(m-1 downto 0);
```

```
  else
    div<=-par_rez(m-1 downto 0);
  end if;
 end process;
end divider_arch;
```

::

Slip-calculation-register.vhd

::

```
-- ***********************************************************

--*************************************************************

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_signed.all;
use work.imp_pack.all;


entity dat2 is

    port (clk,reset:in std_logic;
     in2: in STD_LOGIC_VECTOR (10 downto 0);--10
     out2: out STD_LOGIC_VECTOR(10 downto 0)--10
      );
end dat2;

architecture dat2_arch of dat2 is
 signal out2_par: STD_LOGIC_VECTOR(10 downto 0);  --10

begin


 out2_par<=in2;
-- out1<(in1 & "00000")+in1;
-- Multiplication by 1/Tr = 100e-6/0.024 = 0.0041=0.0000000100001
 process(clk,reset,out2_par)
  variable rez: STD_LOGIC_VECTOR(24 downto 0);
 begin
      if reset ='1' then
       out2<=(others=>'0');
      elsif clk='1' and clk'event then
          rez := rez + (out2_par & zeroes(1))+(out2_par & zeroes(6));

           out2<=rez(24 downto 14);
      end if;
 end process;
end dat2_arch;
```

```
-- End Clark transformation
```

Tetha-calculation.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;
use ieee.std_logic_arith.all;
use work.imp_pack.all;

entity tetaa is
port (clk,reset:in std_logic;
        speed:in std_logic_vector (10 downto 0);
        slip:in std_logic_vector (10 downto 0);
        teta:out std_logic_vector (10 downto 0));
end tetaa;
architecture behavioral of tetaa is
signal x:std_logic_vector (10 downto 0);
signal x1:std_logic_vector (10 downto 0);
signal int_teta:std_logic_vector (10 downto 0);
 begin
         x<=(slip+speed);
         x1<=shl(x,"0100")+shl(x,"0010");   -- x multiplied by constant K_speed =20 see
report III
   process (clk,reset,int_teta)
    begin
       if reset='1' then
          int_teta<=(others=>'0');
      elsif clk='1' and clk'event then
      int_teta<=int_teta+x1;
      end if;
   teta<=int_teta;
 end process;
end behavioral;
```

Control-unit-1.vhd

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_unsigned.ALL;
ENTITY ctrm1 IS
```

```
 PORT( ready1 : in std_logic;
       startP:out std_logic
      --startp:out std_logic
       );
END ctrm1;

ARCHITECTURE ctrm1_arch OF ctrm1 IS
begin
 process (ready1)
  begin
  --**********************
         --if (clk='1'and clk'event ) then
            if ready1='1' then
                 startp<='0';
                 --starta<='1';
                 else
                 startp<='1';
                 --starta<='0';
            end if;
      -- end if;
  end process;
END ctrm1_arch;
```

PID-Controller.vhd

```
 library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;
use ieee.std_logic_arith.all;
--use work.imp_pack.all;
-- ********  This design require a 2400 gates to implement ************
-- ***************       PID Controller       ************************
entity speed_regulator is
port ( readys,reset: in std_logic;
        speed:in std_logic_vector (10 downto 0); --11
        isq:in std_logic_vector (10 downto 0); --11
        isd:in std_logic_vector (10 downto 0); --11
        im:in std_logic_vector (10 downto 0); --11
        imref:in std_logic_vector (10 downto 0); --11
        refspeed:in std_logic_vector (10 downto 0);  --11
        isdref:out std_logic_vector (10 downto 0); --11
        vsdref:out std_logic_vector (10 downto 0); --11
        isqref:out std_logic_vector (10 downto 0);    -- it was 21
        vsqref:out std_logic_vector (10 downto 0));  -- it was 21
end speed_regulator;
architecture behavioral of speed_regulator is
          constant q0:std_logic_vector (3 downto 0):="0001";  -- Propotional Controller
          constant q1:std_logic_vector (3 downto 0):="0001";  -- Integral Controller
          constant q2:std_logic_vector (3 downto 0):="0000";  -- Derivative controller
```

```
        constant qc0:std_logic_vector (3 downto 0):="0001";  -- Propotional Controller
        constant qc1:std_logic_vector (3 downto 0):="0001";  -- Integral Controller
        constant qc2:std_logic_vector (3 downto 0):="0000";  -- Derivative controller
        constant qf0:std_logic_vector (3 downto 0):="0001";  -- Propotional Controller
        constant qf1:std_logic_vector (3 downto 0):="0001";  -- Integral Controller
        constant qf2:std_logic_vector (3 downto 0):="0000";  -- Derivative controller

        constant qcd0:std_logic_vector (3 downto 0):="0001";  -- Propotional Controller
        constant qcd1:std_logic_vector (3 downto 0):="0001";  -- Integral Controller
        constant qcd2:std_logic_vector (3 downto 0):="0000";  -- Derivative controller
begin
--  process (clk,reset,speed,isq,isd,im,imref,refspeed)
  process (readys,reset,speed,isq,isd,im,imref,refspeed)
        variable u0:std_logic_vector (10 downto 0);
        variable u1:std_logic_vector (10 downto 0);
        variable u2:std_logic_vector (10 downto 0);
        variable u3:std_logic_vector (10 downto 0);
        variable nspeed:std_logic_vector (10 downto 0);
        variable dif:std_logic_vector (10 downto 0);
        variable isqref1:std_logic_vector (10 downto 0);
        variable uc0:std_logic_vector (10 downto 0);
        variable uc1:std_logic_vector (10 downto 0);
        variable uc2:std_logic_vector (10 downto 0);
        variable uc3:std_logic_vector (10 downto 0);
        variable nisq:std_logic_vector (10 downto 0);
        variable errorq:std_logic_vector (10 downto 0);
        variable vsqref1:std_logic_vector (10 downto 0);
        variable uf0:std_logic_vector (10 downto 0);
        variable uf1:std_logic_vector (10 downto 0);
        variable uf2:std_logic_vector (10 downto 0);
        variable uf3:std_logic_vector (10 downto 0);
        variable nim:std_logic_vector (10 downto 0);
        variable erroflu:std_logic_vector (10 downto 0);
        variable isdref1:std_logic_vector (10 downto 0);
        variable ucd0:std_logic_vector (10 downto 0);
        variable ucd1:std_logic_vector (10 downto 0);
        variable ucd2:std_logic_vector (10 downto 0);
        variable ucd3:std_logic_vector (10 downto 0);
        variable nisd:std_logic_vector (10 downto 0);
        variable errord:std_logic_vector (10 downto 0);
        variable vsdref1:std_logic_vector (10 downto 0);
begin
 if reset='1' then
    isqref<=(others =>'0');
    isqref1:=(others =>'0');
    u0:=(others=>'0');
    u1:=(others=>'0');
    u2:=(others=>'0');
    u3:=(others=>'0');
```

```
dif:=(others=>'0');
nspeed:=(others=>'0');
vsqref<=(others =>'0');
vsqref1:=(others =>'0');
uc0:=(others=>'0');
uc1:=(others=>'0');
   uc2:=(others=>'0');
   uc3:=(others=>'0');
   errorq:=(others=>'0');
   nisq:=(others=>'0');
   isdref<=(others =>'0');
   isdref1:=(others =>'0');
   uf0:=(others=>'0');
   uf1:=(others=>'0');
   uf2:=(others=>'0');
   uf3:=(others=>'0');
   erroflu:=(others=>'0');
   nim:=(others=>'0');
   vsdref<=(others =>'0');
   vsdref1:=(others =>'0');
   ucd0:=(others=>'0');
   ucd1:=(others=>'0');
   ucd2:=(others=>'0');
   ucd3:=(others=>'0');
   errord:=(others=>'0');
        nisd:=(others=>'0');

--   elsif (clk='1' and clk'event) then
  elsif (readys='1' and readys'event) then
-- ********************  PID speed Controller  Kp=1  Ki=1  Kd=0
**********************
  --  if nspeed /= speed or nisq /= isq then
--   if readys='1' then
          nspeed:=speed;
          dif:=refspeed-nspeed;
          isqref1:=dif+u3;
          u0:="00000000000";
          u2:=dif+isqref1+u1;
          u1:=u0;
          u3:=u2;
            isqref<=isqref1;
  -- end if;
-- ********************  PID currentq Controller  Kp=1  Ki=1  Kd=0
**********************
  -- if nisq /= isq then
      nisq:=isq;
          errorq:=isqref1-isq;
          vsqref1:=errorq+uc3;
          uc0:="00000000000";
```

```
            uc2:=erroiq+vsqref1+uc1;
            uc1:=uc0;
            uc3:=uc2;
         vsqref<=vsqref1;
  --   end if;
-- ********************** PID Flux controller
**********************************
   -- if nim /= im or nisd /= isd then
        nim:=im;
           erroflu:=imref-im;
           isdref1:=erroflu+uf3;
           uf0:="00000000000";
           uf2:=erroflu+isdref1+uf1;
           uf1:=uf0;
           uf3:=uf2;
        isdref<=isdref1;
    -- end if;

-- ********************  PID currentd Controller  Kp=1  Ki=1  Kd=0
**********************
    -- if nisd /= isd then
        nisd:=isd;
           errord:=isdref1-isd;
           vsdref1:=errord+ucd3;
           ucd0:="00000000000";
           ucd2:=errord+vsdref1+ucd1;
           ucd1:=ucd0;
           ucd3:=ucd2;
        vsdref<=vsdref1;
  --   end if;
--
--    end if;
  end if;
-- **********************    End of PI Controller   ****************************
--end if;
end process;
end behavioral;
```

:::

Sin-Rom.vhd

:::

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_unsigned.ALL;

ENTITY sin_rom IS
 PORT( clk,reset,enable : in std_logic;
   A: IN std_logic_vector(10 DOWNTO 0);-- it was 10
   q:out std_logic_vector(7 downto 0);
```

```
  q1: out std_logic_vector(7 downto 0));

END sin_rom;

ARCHITECTURE sin_rom_arch OF sin_rom IS
SIGNAL DD:STD_LOGIC_VECTOR (7 DOWNTO 0);
SIGNAL DD1:STD_LOGIC_VECTOR (7 DOWNTO 0);
SIGNAL DD2:STD_LOGIC_VECTOR (7 DOWNTO 0);
SIGNAL B:STD_LOGIC_VECTOR (10 DOWNTO 0);  -- it was 10

  TYPE mem_data IS ARRAY (0 TO 255) OF std_logic_vector(7 downto 0);
  constant co:std_logic_vector(7 downto 0):="01000000";
  constant VD: mem_data :=
    (('0','0','0','0','0','0','0','0'), --0/00
       ('0','0','0','0','0','0','1','1'),--
       ('0','0','0','0','0','1','1','0'),--
       ('0','0','0','0','1','0','0','0'),--
       ('0','0','0','0','1','1','0','0'),--0c
       ('0','0','0','0','1','1','1','1'),--0f
       ('0','0','0','1','0','0','1','1'),--13
       ('0','0','0','1','0','1','1','0'),--16
       ('0','0','0','1','1','0','0','1'), -- <-- 10/19
       ('0','0','0','1','1','1','0','0'),
       ('0','0','0','1','1','1','1','1'),
       ('0','0','1','0','0','0','1','0'),
       ('0','0','1','0','0','1','0','1'),
       ('0','0','1','0','1','0','0','0'),
       ('0','0','1','0','1','0','1','1'),
       ('0','0','1','0','1','1','1','0'),
       ('0','0','1','1','0','0','0','1'), -- <-- 20/31
       ('0','0','1','1','0','0','1','1'),
       ('0','0','1','1','0','1','1','0'),
       ('0','0','1','1','1','0','0','1'),
       ('0','0','1','1','1','1','0','0'),
       ('0','0','1','1','1','1','1','1'),
       ('0','1','0','0','0','0','0','1'),
       ('0','1','0','0','0','1','0','0'),
       ('0','1','0','1','1','0','1','0'), -- <-- 30/5a
       ('0','1','0','1','1','1','0','0'),
       ('0','1','0','1','1','1','1','0'),
       ('0','1','1','0','0','0','0','0'),
       ('0','1','1','0','0','0','1','0'),
       ('0','1','1','0','0','1','0','0'),
       ('0','1','1','0','0','1','1','0'),
       ('0','1','1','0','1','0','0','0'),
       ('0','1','0','0','0','1','1','1'), -- <-- 40/47
       ('0','1','0','0','1','0','0','1'),
       ('0','1','0','0','1','1','0','0'),
       ('0','1','0','0','1','1','1','0'),
```

```
('0','1','0','1','0','0','0','1'),
('0','1','0','1','0','0','1','1'),
('0','1','0','1','0','1','0','1'),
('0','1','0','1','1','0','0','0'),
('0','1','1','0','1','0','1','0'), -- <-- 50/6a
('0','1','1','0','1','0','1','1'),
('0','1','1','0','1','1','0','1'),
('0','1','1','0','1','1','1','1'),
('0','1','1','1','0','0','0','0'),
('0','1','1','1','0','0','0','1'),
('0','1','1','1','0','0','1','1'),
('0','1','1','1','0','1','0','0'),
('0','1','1','1','0','1','0','1'), -- <-- 60/75
('0','1','1','1','0','1','1','0'),
('0','1','1','1','1','0','0','0'),
('0','1','1','1','1','0','0','1'),
('0','1','1','1','1','0','1','0'),
('0','1','1','1','1','0','1','0'),
('0','1','1','1','1','0','1','1'),
('0','1','1','1','1','1','0','0'),
('0','1','1','1','1','1','0','1'),  -- <-- 70/7d
('0','1','1','1','1','1','0','1'),
('0','1','1','1','1','1','1','0'),
('0','1','1','1','1','1','1','0'),
('0','1','1','1','1','1','1','0'),
('0','1','1','1','1','1','1','1'),
('0','1','1','1','1','1','1','1'),
('0','1','1','1','1','1','1','1'),
('0','1','1','1','1','1','1','1'),  -- <-- 80 / 7f
('0','1','1','1','1','1','1','1'),
('0','1','1','1','1','1','1','1'),
('0','1','1','1','1','1','1','1'),
('0','1','1','1','1','1','1','0'),
('0','1','1','1','1','1','1','0'),
('0','1','1','1','1','1','0','1'),
('0','1','1','1','1','1','0','1'),
('0','1','1','1','1','1','0','0'), -- <-- 90/7c
('0','1','1','1','1','1','0','0'),
('0','1','1','1','1','0','1','1'),
('0','1','1','1','1','0','1','0'),
('0','1','1','1','1','0','0','1'),
('0','1','1','1','1','0','0','0'),
('0','1','1','1','0','1','1','1'),
('0','1','1','1','0','1','1','0'),
('0','1','1','1','0','1','0','1'), -- <--a0/75
('0','1','1','1','0','1','0','0'),
('0','1','1','1','0','0','1','0'),
('0','1','1','1','0','0','0','1'),
('0','1','1','1','0','0','0','0'),
```

```
('0','1','1','0','1','1','1','0'),
('0','1','1','0','1','1','0','1'),
('0','1','1','0','1','0','1','1'),
('0','1','1','0','1','0','0','1'), -- <-- b0/69
('0','1','1','0','0','1','1','1'),
('0','1','1','0','0','1','1','0'),
('0','1','1','0','0','1','0','0'),
('0','1','1','0','0','0','1','0'),
('0','1','1','0','0','0','0','0'),
('0','1','0','1','1','1','1','0'),
('0','1','0','1','1','0','1','1'),
('0','1','0','1','1','0','0','1'), -- <-- C0/59
('0','1','0','1','0','1','1','1'),
('0','1','0','1','0','1','0','1'),
('0','1','0','1','0','0','1','0'),
('0','1','0','1','0','0','0','0'),
('0','1','0','0','1','1','1','0'),
('0','1','0','0','1','0','1','1'),
('0','1','0','0','1','0','0','1'),
('0','1','0','0','0','1','1','0'), -- <-- d0/46
('0','1','0','0','0','0','1','1'),
('0','1','0','0','0','0','0','1'),
('0','0','1','1','1','1','1','0'),
('0','0','1','1','1','0','1','1'),
('0','0','1','1','1','0','0','0'),
('0','0','1','1','0','1','1','0'),
('0','0','1','1','0','0','1','1'),
('0','0','1','1','0','0','0','0'), -- <-- e0/30
('0','0','1','0','1','1','0','1'),
('0','0','1','0','1','0','1','0'),
('0','0','1','0','0','1','1','1'),
('0','0','1','0','0','1','0','0'),
('0','0','1','0','0','0','0','1'),
('0','0','0','1','1','1','1','0'),
('0','0','0','1','1','0','1','1'),
('0','0','0','1','1','0','0','0'), -- <-- f0/18
('0','0','0','1','0','1','0','1'),
('0','0','0','1','0','0','1','0'),
('0','0','0','0','1','1','1','1'),
('0','0','0','0','1','1','0','0'),
('0','0','0','0','1','0','0','0'),
('0','0','0','0','0','0','1','0'),
('0','0','0','0','0','0','0','0'), -- <-- 100/00
('1','0','0','0','0','1','0','0'),
('1','0','0','0','0','1','1','1'),
('1','0','0','0','1','0','1','0'),
('1','0','0','0','1','1','0','1'),
('1','0','0','1','0','0','0','0'),
```

```
('1','0','0','1','0','0','1','1'),
('1','0','0','1','0','1','1','0'),
('1','0','0','1','1','0','0','1'),-- <-- 110/99
('1','0','0','1','1','1','0','1'),
('1','0','1','0','0','0','0','0'),
('1','0','1','0','0','0','1','1'),
('1','0','1','0','0','1','1','0'),
('1','0','1','0','1','0','0','1'),
('1','0','1','0','1','0','1','1'),
('1','0','1','0','1','1','1','0'),
('1','0','1','1','0','0','0','1'), -- <-- 120/b1
('1','0','1','1','0','1','0','0'),
('1','0','1','1','0','1','1','1'),
('1','0','1','1','1','0','1','0'),
('1','0','1','1','1','1','0','1'),
('1','0','1','1','1','1','1','1'),
('1','1','0','0','0','0','1','0'),
('1','1','0','0','0','1','0','1'),
('1','1','0','0','0','1','1','1'),-- <-- 130/c7
('1','1','0','0','1','0','1','0'),
('1','1','0','0','1','1','0','0'),
('1','1','0','0','1','1','1','1'),
('1','1','0','1','0','0','0','1'),
('1','1','0','1','0','1','0','0'),
('1','1','0','1','0','1','1','0'),
('1','1','0','1','1','0','0','0'),
('1','1','0','1','1','0','1','0'),-- <-- 140/da
('1','1','0','1','1','1','0','1'),
('1','1','0','1','1','1','1','1'),
('1','1','1','0','0','0','0','1'),
('1','1','1','0','0','0','1','1'),
('1','1','1','0','0','1','0','1'),
('1','1','1','0','0','1','1','1'),
('1','1','1','0','1','0','0','0'),
('1','1','1','0','1','0','1','0'),-- <-- 150/ea
('1','1','1','0','1','1','0','0'),
('1','1','1','0','1','1','0','1'),
('1','1','1','0','1','1','1','1'),
('1','1','1','1','0','0','0','0'),
('1','1','1','1','0','0','1','0'),
('1','1','1','1','0','0','1','1'),
('1','1','1','1','0','1','0','0'),
('1','1','1','1','0','1','1','0'),-- <-- 160/f6
('1','1','1','1','0','1','1','1'),
('1','1','1','1','1','0','0','0'),
('1','1','1','1','1','0','0','1'),
('1','1','1','1','1','0','1','0'),
('1','1','1','1','1','0','1','1'),
('1','1','1','1','1','0','1','1'),
```

('1','1','1','1','1','1','0','0'),
('1','1','1','1','1','1','0','1'),-- <-- 170/fd
('1','1','1','1','1','1','0','1'),
('1','1','1','1','1','1','1','0'),
('1','1','1','1','1','1','1','0'),
('1','1','1','1','1','1','1','0'),
('1','1','1','1','1','1','1','1'),
('1','1','1','1','1','1','1','1'),
('1','1','1','1','1','1','1','1'),
('1','1','1','1','1','1','1','1'),-- <-- 180/ff
('1','1','1','1','1','1','1','1'),
('1','1','1','1','1','1','1','1'),
('1','1','1','1','1','1','1','0'),
('1','1','1','1','1','1','1','0'),
('1','1','1','1','1','1','1','0'),
('1','1','1','1','1','1','0','1'),
('1','1','1','1','1','1','0','1'),
('1','1','1','1','1','1','0','0'),-- <-- 190/fc
('1','1','1','1','1','1','0','0'),
('1','1','1','1','1','0','1','1'),
('1','1','1','1','1','0','1','0'),
('1','1','1','1','1','0','0','1'),
('1','1','1','1','1','0','0','0'),
('1','1','1','1','0','1','1','1'),
('1','1','1','1','0','1','1','0'),
('1','1','1','1','0','1','0','1'),-- <-- 1a0/f5
('1','1','1','1','0','0','1','1'),
('1','1','1','1','0','0','1','0'),
('1','1','1','1','0','0','0','1'),
('1','1','1','0','1','1','1','1'),
('1','1','1','0','1','1','1','0'),
('1','1','1','0','1','1','0','0'),
('1','1','1','0','1','0','1','0'),
('1','1','1','0','1','0','0','1'),-- <-- 1b0/e9
('1','1','1','0','0','1','1','1'),
('1','1','1','0','0','1','0','1'),
('1','1','1','0','0','0','1','1'),
('1','1','1','0','0','0','0','1'),
('1','1','0','1','1','1','1','1'),
('1','1','0','1','1','1','0','1'),
('1','1','0','1','1','0','1','1'),
('1','1','0','1','1','0','0','1'),-- <-- 1c0/d9
('1','1','0','1','0','1','1','0'),
('1','1','0','1','0','1','0','0'),
('1','1','0','1','0','0','1','0'),
('1','1','0','0','1','1','1','1'),
('1','1','0','0','1','1','0','1'),
('1','1','0','0','1','0','1','0'),
('1','1','0','0','1','0','0','0'),

```
('1','1','0','0','0','1','0','1'),-- <-- 1d0/c5
('1','1','0','0','0','0','1','1'),
('1','1','0','0','0','0','0','0'),
('1','0','1','1','1','1','0','1'),
('1','0','1','1','1','0','1','0'),
('1','0','1','1','1','0','0','0'),
('1','0','1','1','0','1','0','1'),
('1','0','1','1','0','0','1','0'),
('1','0','1','0','1','1','1','1'),-- <-- 1e0/af
('1','0','1','0','1','1','0','0'),
('1','0','1','0','1','0','0','1'),
('1','0','1','0','0','1','1','0'),
('1','0','1','0','0','0','1','1'),
('1','0','1','0','0','0','0','0'),
('1','0','0','1','1','1','0','1'),
('1','0','0','1','1','0','1','0'),
('1','0','0','1','0','1','1','1'),
('1','0','0','1','0','1','0','0'),
('1','0','0','1','0','0','0','1'),
('1','0','0','0','1','1','1','0'),
('1','0','0','0','1','0','1','1'),
('1','0','0','0','1','0','0','0'),
('1','0','0','0','0','1','0','1'),
('0','0','0','0','0','0','0','0'));-- <-- 1ff/00
BEGIN
 PROCESS(A,B,DD,DD1)
 begin
    B<=A+co;
    DD<=VD(conv_integer(A));
    DD1<=VD(conv_integer(b));

 END PROCESS;
-- ******************************
-- **    Register for sin-teta
-- ******************************
  reg_process:process (clk,reset)
  begin

  if reset ='1' then
     q<=(others=>'0');          -- to intialize register q and q1 to 0
       q1<=(others=>'0');
    elsif(clk='1'and clk'event ) then
     if enable='1' then
        q<=DD;                  -- register q to represent sin_teta
        q1<= DD1;               -- register q1 to represent cos_teta=(sin_teta+90)
     end if;
  end if;
 end process reg_process ;
```

```
--end behavioral;

--**
END sin_rom_arch;
```

::

Main program of the second part

Main-Program2.vhd

::

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_signed.ALL;
-- ************************************************************************
-- this programme uses readym instead of Bclk in tst prot map U6
--                                        and  teta port map  U13
-- ************************************************************************

ENTITY main511 IS
 PORT(
        vsqref1,vsdref1:in std_logic_vector(10 downto 0);
      qa,qb: in std_logic_vector(7 downto 0);
      Breset,Bclk:in std_logic;
      startp:in std_logic;
      Tr1,Tr2,Tr3,Tr4,Tr5,Tr6:out std_logic);
END main511;

ARCHITECTURE main511_arch of main511 IS
--#########################################################
--component testbench5 IS
--  PORT(Bclk:in std_logic;
--       signal startt:out std_logic;
--       signal ia,ib:out  STD_LOGIC_VECTOR (8 downto 0)
--  );
--END component;
-- #########################################################
component smultiplier is
generic(
         n: in integer:=11;  -- The operand length
         m: in integer :=8;  -- The result length
         step_length:in integer:=1
        );
port(    a:in std_logic_vector (10 downto 0);
         b:in std_logic_vector (7 downto 0);
         prod:out std_logic_vector (18 downto 0);
         clk,start:in std_logic;
         ready:out std_logic);
end component;
component ctrv1 IS
```

```
 PORT( Bclk:in std_logic;
        readyc,readyc1,readyc2,readyc3 : in std_logic;
       ld1:out std_logic;
       ld2:out std_logic;
       ld3:out std_logic;
       ld4:out std_logic);
--       lpwm:out std_logic );
END component;
component reg11 is
 PORT( lv1,Breset : in std_logic;
       in_reg11:in std_logic_vector(11 downto 0);
       out_reg11:out std_logic_vector(11 downto 0));
end component;
component reg12 is
 PORT( lv2,Breset : in std_logic;
       in_reg12:in std_logic_vector(11 downto 0);
       out_reg12:out std_logic_vector(11 downto 0));
end component;
component reg13 is
 PORT( lv3,Breset : in std_logic;
       in_reg13:in std_logic_vector(11 downto 0);
       out_reg13:out std_logic_vector(11 downto 0));
end component;
component reg14 is
 PORT( lv4,Breset : in std_logic;
       in_reg14:in std_logic_vector(11 downto 0);
       out_reg14:out std_logic_vector(11 downto 0));
end component;
component addr1 IS
 PORT( reset:in std_logic;
        out_r1,out_r2,out_r3,out_r4: in std_logic_vector (11 downto 0);
       vsalref:out std_logic_vector(12 downto 0);
       vsberef:out std_logic_vector(12 downto 0)
      );
END component;
component vpwm_test1 is
   port (Bclk,Breset:in std_logic;
     vsalref: in STD_LOGIC_VECTOR(12 downto 0);
     vsberef: in STD_LOGIC_VECTOR(12 downto 0);
     --out_signala,out_signalb,out_signalc: out STD_LOGIC
     T1,T2,T3,T4,T5,T6:out std_logic
     );
end component;

signal readyv,readyv1,readyv2,readyv3:std_logic;
signal lv11,lv12,lv13,lv14:std_logic;
signal lpwm1,s1,s2,s3:std_logic;
signal outrv1,outrv2,outrv3,outrv4,ina,inb,inc:std_logic_vector(11 downto 0);
signal Vsberef1,vsalref1:std_logic_vector(12 downto 0);
```

```
signal  prodz11,prodz12,prodz13,prodz14:std_logic_vector(18 downto 0);
signal  z11,z12,z13,z14:std_logic_vector(11 downto 0);
begin

-- ***************
--Tb:testbench5 port map(Bclk,start,ia,ib);
*************************************************************************
u16:Smultiplier port map(vsqref1,qb,prodz11,Bclk,startp,readyv);
u17:Smultiplier port map(vsdref1,qa,prodz12,Bclk,startp,readyv1);
u18:Smultiplier port map(vsdref1,qb,prodz13,Bclk,startp,readyv2);
u19:Smultiplier port map(vsqref1,qa,prodz14,Bclk,startp,readyv3);
z11<=prodz11(18 downto 7);
z12<=prodz12(18 downto 7);
z13<=prodz13(18 downto 7);
z14<=prodz14(18 downto 7);
u21:ctrv1 port map(Bclk,readyv,readyv1,readyv2,readyv3,lv11,lv12,lv13,lv14);
u22:reg11 port map(lv11,Breset,z11,outrv1);
u23:reg12 port map(lv12,Breset,z12,outrv2);
u24:reg13 port map(lv13,Breset,z13,outrv3);
u25:reg14 port map(lv14,Breset,z14,outrv4);
u26:addr1 port map(Breset,outrv1,outrv2,outrv3,outrv4,vsalref1,Vsberef1);
u27:vpwm_test1 port map(Bclk,Breset,vsalref1,vsberef1,Tr1,Tr2,Tr3,Tr4,Tr5,Tr6);

End main511_arch;
```

Control-unit-2.vhd

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_unsigned.ALL;

ENTITY ctrv1 IS
 PORT( Bclk,readyc,readyc1,readyc2,readyc3 : in std_logic;

       ld1:out std_logic;
       ld2:out std_logic;
       ld3:out std_logic;
       ld4:out std_logic);
--       lpwm:out std_logic);
     -- out_reg1:out std_logic_vector(19 downto 0));

END ctrv1;

ARCHITECTURE ctrv1_arch OF ctrv1 IS
```

```
begin

 process (Bclk,readyc,readyc1,readyc2,readyc3)
  begin
  --*********************
--  if readyc='1' then
          if (Bclk='1'and Bclk'event ) then
           if readyc='1' then
            ld1<='1';
            else
            ld1<='0';
           end if;
           if readyc1='1' then
            ld2<='1';
            else
            ld2<='0';
           end if;
         if readyc2='1' then
            ld3<='1';
            else
            ld3<='0';
           end if;
          if readyc3='1' then
            ld4<='1';
         --lpwm<='0';
            else
            ld4<='0';
            --lpwm<='1';
           end if;
   end if;
  end process;
END ctrv1_arch;
```

::

Reg11, Reg12, Reg13 and Reg14

::

```
 LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_unsigned.ALL;

ENTITY reg11 IS
 PORT( lv1,Breset : in std_logic;
      in_reg11:in std_logic_vector(11 downto 0);
      out_reg11:out std_logic_vector(11 downto 0));

END reg11;

ARCHITECTURE reg11_arch OF reg11 IS
```

```
begin

 process (lv1,Breset)
  begin

  if Breset ='1' then
     out_reg11<=(others=>'0');          -- to intialize register q and q1 to 0
 elsif(lv1='1'and lv1'event ) then
      out_reg11 <= in_reg11;
     end if;

  end process;

END reg11_arch;
.....

 LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_unsigned.ALL;

ENTITY reg12 IS
 PORT( lv2,Breset : in std_logic;
      in_reg12:in std_logic_vector(11 downto 0);
      out_reg12:out std_logic_vector(11 downto 0));
END reg12;

ARCHITECTURE reg12_arch OF reg12 IS

begin

 process (lv2,Breset)
  begin

  if Breset ='1' then
     out_reg12<=(others=>'0');          -- to intialize register q and q1 to 0
     elsif(lv2='1'and lv2'event ) then
     out_reg12 <= in_reg12;
  end if;

  end process;

END reg12_arch;
.....

 LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_unsigned.ALL;
```

```
ENTITY reg13 IS
 PORT( lv3,Breset: in std_logic;
       in_reg13:in std_logic_vector(11 downto 0);
       out_reg13:out std_logic_vector(11 downto 0));
END reg13;

ARCHITECTURE reg13_arch OF reg13 IS

begin

 process (lv3,Breset)
  begin

  if Breset ='1' then
     out_reg13<=(others=>'0');          -- to intialize register q and q1 to 0
     elsif(lv3='1'and lv3'event ) then
     out_reg13 <= in_reg13;
  end if;

  end process;

END reg13_arch;
  ..................................................

 LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_unsigned.ALL;

ENTITY reg14 IS
 PORT( lv4,Breset: in std_logic;
       in_reg14:in std_logic_vector(11 downto 0);
       out_reg14:out std_logic_vector(11 downto 0));
END reg14;

ARCHITECTURE reg14_arch OF reg14 IS

begin

 process (lv4,Breset)
  begin

  if Breset ='1' then
     out_reg14<=(others=>'0');          -- to intialize register q and q1 to 0
     elsif(lv4='1'and lv4'event ) then
     out_reg14 <= in_reg14;
  end if;

  end process;
```

```
END reg14_arch;
```

Addr1.vhd

```
 LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_signed.ALL;
use ieee.std_logic_arith.all;

ENTITY addr1 IS
 PORT( reset:in std_logic;
        out_r1,out_r2,out_r3,out_r4: in std_logic_vector (11 downto 0);
       vsalref:out std_logic_vector(12 downto 0);
       vsberef:out std_logic_vector(12 downto 0)
     );
END addr1;

ARCHITECTURE addr1_arch OF addr1 IS


begin

 process (reset,out_r1,out_r2,out_r3,out_r4)
  begin
   if reset ='1' then
    vsberef<=(others=>'0');
    vsalref<=(others=>'0');
        else
     vsberef <= (out_r1(10) & out_r1) + out_r2;
     vsalref <= (out_r3(10) & out_r3) - out_r4;
   end if;
 end process;

END addr1_arch;
```

PWM-Generation.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_signed.all;
use ieee.std_logic_arith.all;
use work.imp_pack.all;


entity vpwm_test1 is
```

```
--    port (Bclk,lv,Breset:in std_logic;
    port (Bclk,Breset:in std_logic;
      vsalref: in STD_LOGIC_VECTOR(12 downto 0);
      vsberef: in STD_LOGIC_VECTOR(12 downto 0);
      --v1,v2,v3:out STD_LOGIC_VECTOR(13 downto 0);
        --out_signala,out_signalb,out_signalc: out std_logic
      T1,T2,T3,T4,T5,T6:out std_logic
      );
end vpwm_test1;

Architecture vpwm_test1_arch of vpwm_test1 is

 signal dout1,dout2,x1,dout3,dout4,x2,dout5,dout6,x3 :  std_logic;
-- signal out_signala1,out_signala2,out_signala3:std_logic;
 signal reg1,reg2,reg3,reg4,reg5,reg6 : STD_LOGIC_VECTOR(3 downto 0);
 signal out_signala,out_signalb,out_signalc: std_logic ;
 signal vsalpha1: STD_LOGIC_VECTOR(12 downto 0);
 signal vsbeta1: STD_LOGIC_VECTOR(12 downto 0);
 signal vbeta_par: STD_LOGIC_VECTOR(12 downto 0);
 signal vrefa,vrefb,vrefc: STD_LOGIC_VECTOR(12 downto 0);
 constant max:std_logic_vector (11 downto 0):="011111111111";
 constant min:std_logic_vector (11 downto 0):="100000000000";
constant DelayTTr:integer:=30;

begin
  vsbeta1(12)<='0';
  vsbeta1(11 downto 0) <= vsberef(12 downto 1); --    To divide vsberef by 1/2
process(vsalref)
    variable rez: STD_LOGIC_VECTOR(25 downto 0);
    variable Triangl:  std_logic_vector (12 downto 0);
    variable direction:  std_logic;
begin

 rez := vsalref & zeroes(13);--"000000000000";                --zeroes(12);
 rez := rez - (vsalref & zeroes(10));--"000000000");          -- 9 = 12 - 3
 rez := rez - (vsalref & zeroes(6));--"00000");               -- 5 = 12 - 7
 rez := rez - (vsalref & zeroes(3));--;"00");            -- 2 = 12 - 10
 rez := rez - vsalref;                        -- 0 = 12 - 12
 vsalpha1<=rez(25 downto 13);

-- vrefa<=vsberef;
-- vrefb<=vsalpha1-vsbeta1;
-- vrefc<=-vsalpha1-vsbeta1;
-- v1<=vrefa;
-- v2<=vrefb;
-- v3<=vrefc;

end process;
```

```
 vrefa<=vsberef;
 vrefb<=vsalpha1-vsbeta1;
 vrefc<=-vsalpha1-vsbeta1;
process(Bclk,Breset)
    variable rez: STD_LOGIC_VECTOR(25 downto 0);
    variable Triangl:  std_logic_vector (12 downto 0);
    variable direction:  std_logic;
 begin
   if Breset='1'then
    Triangl:="0000000000001";
    direction :='1';
   elsif (Bclk'event and Bclk='1') then
    --if lv='1' then                                    --   <---
       if direction ='1' then
         if Triangl<max then
            Triangl:=Triangl+1;
            else
            Triangl:=Triangl-1;
               direction:='0';
             end if;
            else
            if Triangl>min then
               Triangl:=Triangl-1;
               else
               Triangl:=Triangl+1;
                 direction:='1';
              end if;
           end if;
        if vrefa< Triangl then      -- vr1 coresponds to either va or vb or vc
           out_signala<='0';
           else
          out_signala<='1';
        end if;
       if vrefb< Triangl then       -- vr2 coresponds to either va or vb or vc
           out_signalb<='0';
           else
          out_signalb<='1';
        end if;
        if vrefc< Triangl then      -- vr3 coresponds to either va or vb or vc
           out_signalc<='0';
           else
          out_signalc<='1';
        end if;
  end if;
end process;
-- ------------------------------------------------
process (Bclk,Breset)--,out_signalb,out_signalc)
        variable count: integer range 0 to 31;
        --variable dout1,dout2 :  std_logic;
```

```
        --variable reg1,reg2 : STD_LOGIC_VECTOR(2 downto 0);
    begin
     if Breset ='1' then
       x1<='0';
          x2<='0';
          x3<='0';
       dout1<='0';
          dout2<='0';
          dout3<='0';
          dout4<='0';
          dout5<='0';
          dout6<='0';
         T1<='0';
         T2<='0';
         T3<='0';
         T4<='0';
         T5<='0';
         T6<='0';
         count:=delayTTr;
    elsif Bclk'event and Bclk='1' then
     if count=0 then
       count:=DelayTTr;
-- ++++++++++++++++++++++++++++++++++++++++++++++
        x1 <= not out_signala;
        reg1<= out_signala & reg1(3 downto 1);
          reg2<= x1 & reg2(3 downto 1);
                                --end if;
          dout1<=reg1(0);
          dout2<=reg2(0);
       T1 <= out_signala and dout1;
          T2 <= x1 AND dout2;
-- ++++++++++++++++++++++++++++++++++++++++++++++
         x2 <= not out_signalb;
          reg3<= out_signalb & reg3(3 downto 1);
          reg4<= x2 & reg4(3 downto 1);
          dout3<=reg3(0);
          dout4<=reg4(0);
          T3 <= out_signalb and dout3;
          T4 <= x2 AND dout4;
-- ++++++++++++++++++++++++++++++++++++++++++++++
        x3 <= not out_signalc;
          reg5<= out_signalc & reg5(3 downto 1);
          reg6<= x3 & reg6(3 downto 1);
          dout5<=reg5(0);
          dout6<=reg6(0);
          T5 <= out_signalc and dout5;
          T6 <= x3 AND dout6;
-- ++++++++++++++++++++++++++++++++++++++++++++++
    else
```

```
      count:=count-1;
     end if;
   end if;
  end process;
end vpwm_test1_arch;
```

Appendix C

****
****
********** *VHDL Code* **********
****
****

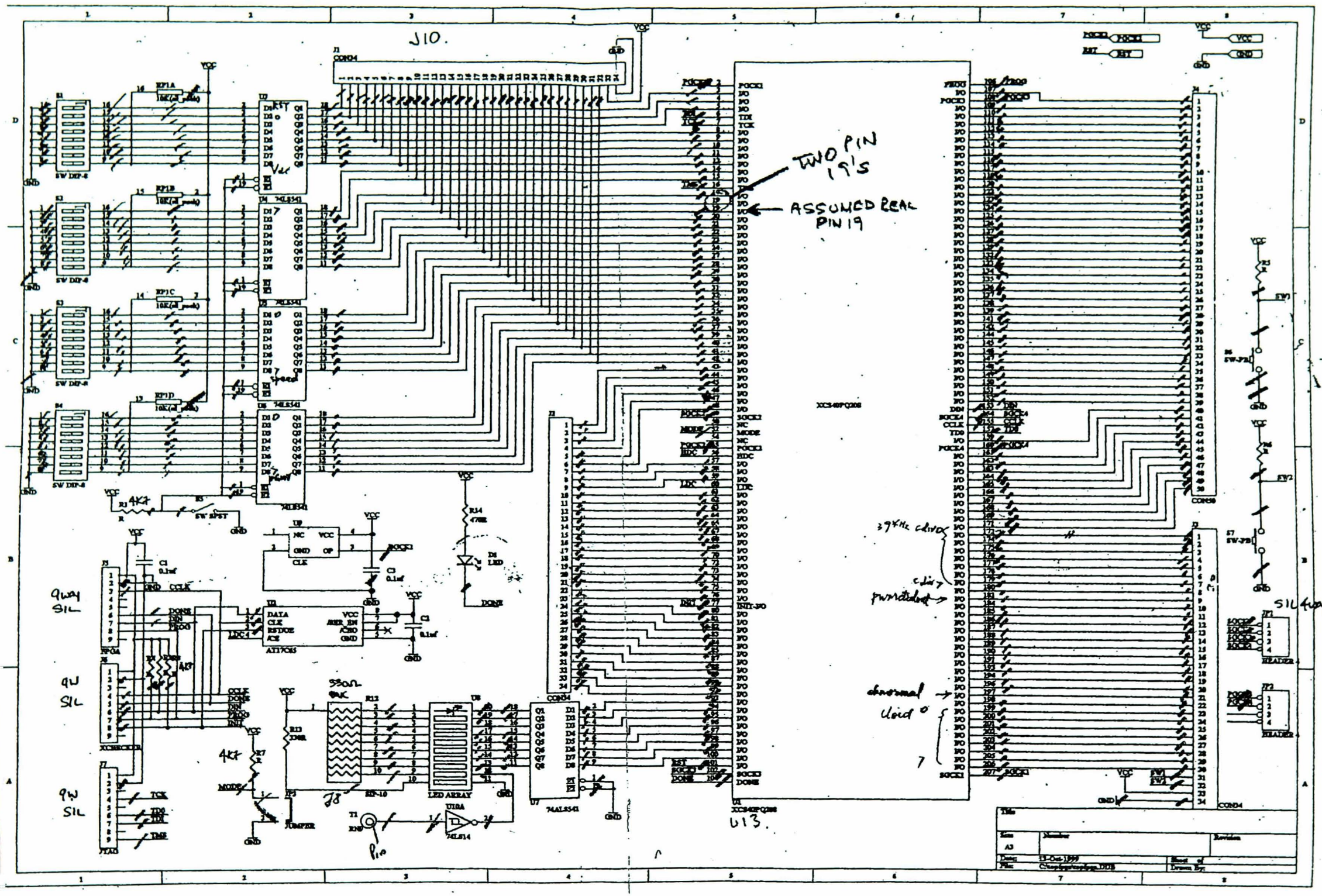
Testbenches and Simulations

****

```
 LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_unsigned.ALL;

ENTITY testbench5 IS
  PORT(clk:in std_logic;
        signal startt:out std_logic;
        signal ia,ib:out  STD_LOGIC_VECTOR (8 downto 0)
        --signal qa,qb:out  STD_LOGIC_VECTOR (7 downto 0)
  );

END testbench5;

ARCHITECTURE testbench5_arch OF testbench5 IS

--begin

  signal counter: integer range 0 to 13000;
  signal counter1: integer range 0 to 1000;
-- *********  These constant represents sin data  s=a ************

  constant s1:STD_LOGIC_VECTOR (8 downto 0):="000000000";-- 0
  constant s2:STD_LOGIC_VECTOR (8 downto 0):="000010100";-- 20
  constant s3:STD_LOGIC_VECTOR (8 downto 0):="000101000";-- 40
  constant s4:STD_LOGIC_VECTOR (8 downto 0):="000111100";-- 60
  constant s5:STD_LOGIC_VECTOR (8 downto 0):="000101000";-- 40
  constant s6:STD_LOGIC_VECTOR (8 downto 0):="000010100";-- 20
  constant s7:STD_LOGIC_VECTOR (8 downto 0):="000000000";-- 0
  constant s8:STD_LOGIC_VECTOR (8 downto 0):="111101100";--  (-20)
  constant s9:STD_LOGIC_VECTOR (8 downto 0):="111011000";--  (-40)
  constant s10:STD_LOGIC_VECTOR (8 downto 0):="111000100";-- (-60)
  constant s11:STD_LOGIC_VECTOR (8 downto 0):="111011000";--  (-40)
  constant s12:STD_LOGIC_VECTOR (8 downto 0):="111101100";--  (-20)
  constant s13:STD_LOGIC_VECTOR (8 downto 0):="000000000";-- 0

-- *********  These constant represents cos data  c=b ************
```

```
 constant c1:STD_LOGIC_VECTOR (8 downto 0):="000111100";-- 60
 constant c2:STD_LOGIC_VECTOR (8 downto 0):="000101000";-- 40
 constant c3:STD_LOGIC_VECTOR (8 downto 0):="000010100";-- 20
 constant c4:STD_LOGIC_VECTOR (8 downto 0):="000000000";-- 0
 constant c5:STD_LOGIC_VECTOR (8 downto 0):="111101100";-- (-20)
 constant c6:STD_LOGIC_VECTOR (8 downto 0):="111011000";-- (-40)
 constant c7:STD_LOGIC_VECTOR (8 downto 0):="111000100";-- (-60)
 constant c8:STD_LOGIC_VECTOR (8 downto 0):="111011000";-- (-40)
 constant c9:STD_LOGIC_VECTOR (8 downto 0):="111101100";-- (-20)
 constant c10:STD_LOGIC_VECTOR (8 downto 0):="000000000";-- 0
 constant c11:STD_LOGIC_VECTOR (8 downto 0):="000010100";-- 20
 constant c12:STD_LOGIC_VECTOR (8 downto 0):="000101000";-- 40
 constant c13:STD_LOGIC_VECTOR (8 downto 0):="000111100";-- 60
--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++=
begin
process (clk,counter)
  begin
  if clk'event and clk='1' then
      counter <=counter+1;
   end if;
    if counter <1000 and counter >=0 then
      ia<=s1;
      ib<=c1;
     -- qa<=as1;
     -- qb<=ac1;
      elsif counter <2000 and counter >1000 then
      ia<=s2;
      ib<=c2;

     elsif counter <3000 and counter >2000 then
      ia<=s3;
      ib<=c3;

     elsif counter <4000 and counter >3000 then
      ia<=s4;
      ib<=c4;

    elsif counter <5000 and counter >4000 then
      ia<=s5;
      ib<=c5;

   elsif counter <6000 and counter >5000 then
      ia<=s6;
      ib<=c6;

   elsif counter <7000 and counter >6000 then
```

```
   ia<=s7;
   ib<=c7;

elsif counter <8000 and counter >7000 then
   ia<=s8;
   ib<=c8;

elsif counter <9000 and counter >8000 then
   ia<=s9;
   ib<=c9;

elsif counter <10000 and counter >9000 then
   ia<=s10;
   ib<=c10;

elsif counter <11000 and counter >10000 then
   ia<=s11;
   ib<=c11;

elsif counter <12000 and counter >11000 then
   ia<=s12;
   ib<=c12;

elsif counter <13000 and counter >12000 then
   ia<=s13;
   ib<=c13;

  elsif counter>13000 then
        counter<=0;
  end if;

end process;

process (clk,counter1)
 begin
 if clk'event and clk='1' then
      counter1 <=counter1+1;
  end if;
 if counter1>= 950 and counter1 <1000 then --and readyt='1' then
     startt<='1';
     else
     startt<='0';
 end if;
 if counter1>1000 then
   counter1<=0;
 end if;
end process;

END testbench5_arch;
```


J10.
TWO PIN 19's
ASSUMED REAL PIN 19
37kHz clock
clock
power down
channel Used 0
7
4K7
330Ω
9way SIL
9W SIL
9W SIL
SW DIP-8
SW-PB
SW-SPST
LED ARRAY
74LS14
74ALS541
HEADER 4
CON64
CON68
Title
Size
A3
Number
Revision
Date:
13-Oct-1999
Sheet of
File:
Drawn By:

Induction Machine Model.

The continuous-time electromechanical model of an induction machine is of fifth order and non-linear [**Error! Bookmark not defined.**] and can be represented as:

Electrical Model

$$\frac{di_{ds}}{dt} = (R_s L_r i_{ds} + \omega_r L_m^2 i_{qs} - R_r L_m i_{dr} - \omega_r L_r L_m i_{qr} - L_r v_{ds}) / a_0$$

$$\frac{di_{qs}}{dt} = (\omega_r L_m^2 i_{ds} + R_s L_r i_{qs} + \omega_r L_r L_m i_{dr} - R_r L_m i_{qr} - L_r v_{qs}) / a_0$$

$$\frac{di_{dr}}{dt} = -(R_s L_m i_{ds} - \omega_r L_m L_s i_{qs} - R_r L_s i_{dr} - \omega_r L_r L_s i_{qr} - L_m v_{ds}) / a_0$$

$$\frac{di_{qr}}{dt} = -(\omega_r L_m L_s i_{ds} + R_s L_m i_{qs} + \omega_r L_r L_s i_{dr} - R_r L_s i_{qr} - L_m v_{qs}) / a_0$$

$$T_e = \frac{3}{2} L_m (i_{qs} i_{dr} - i_{ds} i_{qr})$$

where $a_0 = (L_m * L_m - L_r * L_s)$;

Mechanical Model

To define the behaviour of the induction motor, the system motion equation needs to be integrated with the *dq-axis* equations. The load torque of the motor may be due to friction, windage, acceleration and mechanical work. Neglecting the coulomb and static friction components, the torque due to friction is approximated to $T_F = B\omega_r$, viscous friction, where B is a constant for the system and ω_r is the rotor speed. The windage torque is combined with the viscous torque and is given by $T_{F+W} = D\omega_r$ where *D* is the damping constant. The torque component required to accelerate the system is expressed as:

$T_J = J\frac{d\omega_r}{dt}$, where J is the rotational inertia of the system in $Kg.m^2$.

The load torque T_l,, is the mechanical work required and can be expressed as a function of ω_r. Therefore, the developed electromagnetic torque T is given by:

$$T_e = J\frac{d\omega_r}{dt} + D\omega_r + T_l$$

$$\omega_r = \frac{1}{J}\int (T_e - T_l)dt.$$

where T_e = the electric developed torque.

Sine Look-up Table EPROM Data

0000/00;0001/01;0002/03;0003/05;0004/06;0005/06;0006/08;0007/0b;0008/0c;
0009/0e;000a/0f;000b/11;000c/13;000d/14;000e/16;000f/17;0010/19;0011/1a;
0012/1c;0013/1d;0014/1f;0015/20;0016/22;0017/23;0018/25;0019/26;001a/28;
001b/29;001c/2b;001d/2c;001e/2e;001f/2f;0020/31;0021/32;0022/33;0023/35;
0024/36;0025/38;0026/39;0027/3a;0028/3c;0029/3d;002a/3f;002b/40;002c/41;
002d/43;002e/44;002f/45;0030/5a;0031/5b;0032/5c;0033/5d;0034/5e;0035/5f;
0036/60;0037/61;0038/62;0039/63;003a/64;003b/65;003c/66;003d/67;003e/68;
003f/69;0040/47;0041/48;0042/49;0043/4a;0044/4c;0045/4d;0046/4e;0047/4f;
0048/51;0049/52;004a/53;004b/54;004c/55;004d/56;004e/58;004f/59;0050/6a;
0051/6a;0052/6b;0053/6c;0054/6d;0055/6e;0056/6f;0057/6f;0058/70;0059/71;
005a/71;005b/72;005c/73;005d/73;005e/74;005f/75;0060/75;0061/76;0062/76;
0063/77;0064/78;0065/78;0066/79;0067/79;0068/7a;0069/7a;006a/7a;006b/7b;
006c/7b;006d/7c;006e/7c;006f/7c;0070/7d;0071/7d;0072/7d;0073/7d;0074/7e;
0075/7e;0076/7e;0077/7e;0078/7e;0079/7e;007a/7f;007b/7f;007c/7f;007d/7f;
007e/7f;007f/7f;0080/7f;0081/7f;0082/7f;0083/7f;0084/7f;0085/7f;0086/7f;
0087/7e;0088/7e;0089/7e;008a/7e;008b/7e;008c/7d;008d/7d;008e/7d;008f/7d;
0090/7c;0091/7c;0092/7c;0093/7b;0094/7b;0095/7b;0096/7a;0097/7a;0098/79;
0099/79;009a/78;009b/78;009c/77;009d/77;009e/76;009f/76;00a0/75;00a1/74;
00a2/74;00a3/73;00a4/72;00a5/72;00a6/71;00a7/70;00a8/70;00a9/6f;00aa/6e;
00ab/6d;00ac/6d;00ad/6c;00ae/6b;00af/6a;00b0/69;00b1/68;00b2/67;00b3/67;
00b4/66;00b5/65;00b6/64;00b7/63;00b8/62;00b9/61;00ba/60;00bb/5f;00bc/5e;
00bd/5d;00be/5b;00bf/5a;00c0/59;00c1/58;00c2/57;00c3/56;00c4/55;00c5/54;
00c6/52;00c7/51;00c8/50;00c9/4f;00ca/4e;00cb/4c;00cc/4b;00cd/4a;00ce/49;
00cf/47;00d0/46;00d1/45;00d2/43;00d3/42;00d4/41;00d5/3f;00d6/3e;00d7/3d;
00d8/3b;00d9/3a;00da/38;00db/37;00dc/36;00dd/34;00de/33;00df/31;00e0/30;
00e1/2e;00e2/2d;00e3/2b;00e4/2a;00e5/29;00e6/27;00e7/26;00e8/24;00e9/23;
00ea/21;00eb/20;00ec/1e;00ed/1d;00ee/1b;00ef/19;00f0/18;00f1/16;00f2/15;
00f3/13;00f4/12;00f5/10;00f6/0f;00f7/0d;00f8/0c;00f9/0a;00fa/08;00fb/07;
00fc/05;00fd/04;00fe/02;00ff/00;0100/00;0101/82;0102/84;0103/85;0104/87;
0105/88;0106/8a;0107/8c;0108/8d;0109/8f;010a/90;010b/92;010c/93;010d/95;
010e/96;010f/98;0110/99;0111/9b;0112/9d;0113/9e;0114/a0;0115/a1;0116/a3;
0117/a4;0118/a6;0119/a7;011a/a9;011b/aa;011c/ab;011d/ad;011e/ae;011f/b0;
0120/b1;0121/b3;0122/b4;0123/b6;0124/b7;0125/b8;0126/ba;0127/bb;0128/bd;
0129/be;012a/bf;012b/c1;012c/c2;012d/c3;012e/c5;012f/c6;0130/c7;0131/c9;
0132/ca;0133/cb;0134/cc;0135/ce;0136/cf;0137/d0;0138/d1;0139/d2;013a/d4;
013b/d5;013c/d6;013d/d7;013e/d8;013f/d9;0140/da;0141/db;0142/dd;0143/de;
0144/df;0145/e0;0146/e1;0147/e2;0148/e3;0149/e4;014a/e5;014b/e6;014c/e7;
014d/e7;014e/e8;014f/e9;0150/ea;0151/eb;0152/ec;0153/ed;0154/ed;0155/ee;
0156/ef;0157/f0;0158/f0;0159/f1;015a/f2;015b/f2;015c/f3;015d/f4;015e/f4;

Sine Look-up Table EPROM Data

015f/f5;0160/f6;0161/f6;0162/f7;0163/f7;0164/f8;0165/f8;0166/f9;0167/f9;
0168/fa;0169/fa;016a/fb;016b/fb;016c/fb;016d/fc;016e/fc;016f/fc;0170/fd;
0171/fd;0172/fd;0173/fd;0174/fe;0175/fe;0176/fe;0177/fe;0178/fe;0179/ff;
017a/ff;017b/ff;017c/ff;017d/ff;017e/ff;017f/ff;0180/ff;0181/ff;0182/ff;
0183/ff;0184/ff;0185/ff;0186/fe;0187/fe;0188/fe;0189/fe;018a/fe;018b/fe;
018c/fd;018d/fd;018e/fd;018f/fd;0190/fc;0191/fc;0192/fc;0193/fb;0194/fb;
0195/fa;0196/fa;0197/fa;0198/f9;0199/f9;019a/f8;019b/f8;019c/f7;019d/f6;
019e/f6;019f/f5;01a0/f5;01a1/f4;01a2/f3;01a3/f3;01a4/f2;01a5/f1;01a6/f1;
01a7/f0;01a8/ef;01a9/ef;01aa/ee;01ab/ed;01ac/ec;01ad/eb;01ae/ea;01af/ea;
01b0/e9;01b1/e8;01b2/e7;01b3/e6;01b4/e5;01b5/e4;01b6/e3;01b7/e2;01b8/e1;
01b9/e0;01ba/df;01bb/de;01bc/dd;01bd/dc;01be/db;01bf/da;01c0/d9;01c1/d8;
01c2/d6;01c3/d5;01c4/d4;01c5/d3;01c6/d2;01c7/d1;01c8/cf;01c9/ce;01ca/cd;
01cb/cc;01cc/ca;01cd/c9;01ce/c8;01cf/c7;01d0/c5;01d1/c4;01d2/c3;01d3/c1;
01d4/c0;01d5/bf;01d6/bd;01d7/bc;01d8/ba;01d9/b9;01da/b8;01db/b6;00dc/b5;
01dd/b3;01de/b2;01df/b1;01e0/af;01e1/ae;01e2/ac;01e3/ab;01e4/a9;01e5/a8;
01e6/a6;01e7/a5;01e8/a3;01e9/a2;01ea/a0;01eb/9f;01ec/9d;01ed/9c;01ee/9a;
01ef/99;01f0/97;01f1/96;01f2/94;01f3/93;01f4/91;01f5/8f;01f6/8e;01f7/8c;
01f8/8b;01f9/89;01fa/88;01fb/86;01fc/85;01fd/83;01fe/81;01ff/00;